文本复制检测报告单(全文标明引文)

№:ADBD2021R_2021051120175545730709686 4 检测时间:2021-05-11 20:17:55

检测文献: 工业物联网数据管理信息系统与终端硬件设计
作者: 马思清
检测范围:
中国学术期刊网络出版总库
中国博士学位论文全文数据库/中国优秀硕士学位论文全文数据库
中国重要会议论文全文数据库
中国重要报纸全文数据库
中国专利全文数据库
图书资源
优先出版文献库
大学生论文联合比对库
互联网资源(包含贴吧等论坛资源)
英文数据库(涵盖期刊、博硕、会议的英文数据以及德国Springer、英国Taylor&Francis 期刊数据库等)
港澳台学术文献库
互联网文档资源
源代码库
CNKI大成编客-原创作品库
个人比对库
时间范围: 1900-01-01至2021-05-11

检测结果

去除本人文献复制比: 5.8% 跨语言检测结果: 0%

去除引用文献复制比: 5.8% 总文字复制比: 5.8%

单篇最大文字复制比: 2.8%(源代码(/* USER CODE BEGIN Header */ /** **********...))

重复字数: [3937]	总字数: [67975]	单篇最大重复字数: [1874]
总段落数: [6]	前部重合字数: [406]	疑似段落最大重合字数: [2435]
疑似段落数: [5]	后部重合字数: [3531]	疑似段落最小重合字数: [29]

指 标: ☐疑似剽窃观点 ☑疑似剽窃文字表述 ☐疑似整体剽窃 ☐过度引用

表 格: 0 公 式: 没有公式 疑似文字的图片: 0 脚注与尾注: 0

0.3%(29)	0.3%(29)	工业物联网数据管理信息系统与终端硬件设计_第1部分(总9580字)
4.2%(411)	4.2%(411)	工业物联网数据管理信息系统与终端硬件设计_第2部分(总9799字)
3.5%(318)	3.5%(318)	工业物联网数据管理信息系统与终端硬件设计_第3部分(总9026字)
15.3%(2435)	15.3%(2435)	工业物联网数据管理信息系统与终端硬件设计_第4部分(总15869字)
4.6%(744)	4.6%(744)	工业物联网数据管理信息系统与终端硬件设计_第5部分(总16161字)
0%(0)	0%(0)	工业物联网数据管理信息系统与终端硬件设计_第6部分(总7540字)

(注释: 无问题部分 文字复制部分 引用部分)

指导教师审查结果

指导教师: 虞亚军
审阅结果:
审阅意见: 指导老师未填写审阅意见

1. 工业物联网数据管理信息系统与终端硬件设计_第1部分

总字数: 9580

相似文献列表

去除本人文献复制比: 0.3%(29) 文字复制比: 0.3%(29) 疑似剽窃观点: (0)

1	EDA技术在数字电子技术实验中的应用	0.3%(29) 是否引证: 否

原文内容

分类号编号

U D C 密级

本科生毕业设计（论文）

题目： 工业物联网数据管理信息系统与终端硬件设计

姓名： 马思清

学号： 11712610

系别： 电子与电气工程系

专业： 信息工程

指导教师： 虞亚军

2021 年 5月 10日

诚信承诺书

1.本人郑重承诺所呈交的毕业设计（论文），是在导师的指导下，独立进行研究工作所取得的成果，所有数据、图片资料均真实可靠。

2.除文中已经注明引用的内容外，本论文不包含任何其他人或集体已经发表或撰写过的作品或成果。对本论文的研究作出重要贡献的个人和集体，均已在文中以明确的方式标明。

3.本人承诺在毕业论文（设计）选题和研究内容过程中没有抄袭他人研究成果和伪造相关数据等行为。

4.在毕业论文（设计）中对侵犯任何方面知识产权的行为，由本人承担相应的法律责任。

作者签名：

年月日

工业物联网数据管理信息系统与终端硬件设计

马思清

（电子与电气工程系指导教师：虞亚军）

[摘要]： 本设计是关于工业企业物联网系统的一套完整搭建方案，涉及设备数据采集，信号处理，信息分析以及信息的可视化应用，涵盖了物联网信息收集、处理、展示全流程。本方案的具体内容包括一个数据采集硬件设备，一个联接管理软件系统，与一个用户端应用程序。数据采集终端按照工业标准设计，可采集工业设备的各种数据，实现了多种模拟量与传感器信号的接入，并搭载最新的窄带物联网（NB-IoT）技术，有着良好的无线通信性能。联接管理系统是一个运行在云服务器上的程序，收集各个硬件终端通过互联网上报的消息数据，对其中的信号进行预处理和分析，转移和储存原数据及分析结果，并监控各个硬件终端的工作状态。用户端应用程序基于WEB技术设计，实现了多种数据可视化展示方式，支持用户异地远程监控设备工作情况。本工业物联网搭建方案结合了电子技术，信息处理技术和计算机技术，解决了本地设备数据难以联网所形成的信息孤岛问题，实现了数据的多样化采集，实现了工业设备的云监控和工业数据的云储存。此外，本系统为用户同时提供了软件和硬件解决方案，能一次性部署到位，快速上线使用，解决了因不同厂商硬件设备、云服务、可视化平台标准不一，难以协同开发的问题，有很好的实用价值和发展前景。

[ABSTRACT]: This design is a complete set of building scheme for industrial enterprise IOT system, involving equipment data acquisition, signal processing, information analysis and information visualization application, covering the whole process of IOT information collection, processing and display. This system includes a data collection device, a connection management software system, and a WEB client application. The data collection hardware is designed according to industrial standards, which can collect various data from industrial equipment and realize access to various analog and sensor signals, and equipped with the latest Narrowband Internet of Things (NB-IoT) communication technology, which has good wireless communication performance. The connection management system is a program running on a cloud server that collects message data reported by each hardware terminal through the Internet, pre-processes and analyzes the signals collected, dumps the extracted information, and monitors the working status of each hardware terminal. The user-side application interface is designed based on WEB technology, which realizes data visualization display in various ways and allows users to remotely monitor the working condition of the equipment off-site. This industrial IoT system combines electronic technology, information processing technology and computer technology, solving the problem of information island formed by the difficulty of networking local equipment data, realizing cloud monitoring of industrial equipment and cloud storage of industrial data. Besides, this system combines software and hardware devices together, and which be deployed at one time, solving the problem of different standards of hardware equipment, cloud services and visualization platforms of different manufacturers, that make it difficult to develop in cooperation. In summary, this industrial IoT system has a good practical value and development prospects.

[关键词]：物联网; 智能硬件; 云服务

目录

1. 工业物联网系统介绍

1.1 研发背景

工业为自身和国民经济其他各个部门提供原材料、燃料和动力，是国民经济的重要支柱，在世界各国国民经济中起着主导作用。无论是能源采掘工业、原材料工业、或是加工工业，都大批量的使用机械电子设备来协助生产。提高设备的工作效率，优化生产工艺和生产流程是工业企业实现提升质量、降低成本、增加效益的根本手段。为了实现对生产过程更高效的把握，自20世纪以来，越来越多的精益生产模式被人们所使用，人们希望通过技术手段加强对设备的监控管理，来高效控制生产过程；希望采集并分析更多的生产过程数据来提升和优化设备的工作水平。

为更好的控制和优化生产过程，我们需要实时的获取尽可能多的生产数据。在信息时代之前，人们通过编写各种纸质的报表来手工记录，随着电子技术和计算机技术的不断发展，许多生产设备都实现电子化或数字化控制，人们运用了各种传感器来收集信息，采集到的数据以电子信号的形式传输并储存在工厂的计算机内，用于展示和分析。这一进程被称为工业生产信息的数字化管理。

但仍有部分中低端产能企业未能实现数字化管理，这些企业规模有限，生产产品简单，自动化设备数量少；或是因为存在必要的人工工序、不适合大规模机械化加工，自动化水平低，缺乏数字化管理所需要的基本硬件基础。如建材加工厂、电镀厂

、包装制品厂、喷漆厂等，多为产能有限、布局分散的小微企业，其中生产设备如吊装设备、搅拌设备、燃气锅炉、各式压力容器等，都必须由工人进行操作。这类设备的复杂程度和自动化水平较低，都不具备基本的编程或联网能力。当大数据和智能制造时代来临，这些企业迫切需要一种解决方案，在不变动传统生产设备、不破坏原有生产秩序的前提下实现智能化和数字化管理，以享受信息产业高速发展所带来的红利。

本文所介绍的工业物联网数据管理信息系统与终端硬件（以下统称工业物联网系统）正是在这种背景下开发，其设计目标是利用物联网技术来协助传统工业企业实现数字化和智能化管理，其解决方案是设计一套软硬件系统来收集传统工业设备的运行信息，赋予其联网能力，并提供与之配套的数据分析和数据可视化系统。工业物联网系统面向传统机电设备和孤立的传感器单元设计了一种通用的数据采集模块，同时依靠模块的网络通信功能，实现设备与设备间的互联互通，打破各个生产设备间的数据壁垒，在客户端汇聚海量的设备数据。

依靠上述功能，企业可以以数据为导向有效优化生产过程，高效监管生产设备，获取更多有价值的生产信息，实现精细化管理。此外，工业物联网系统还会融合先进的通信技术以及更强大的云端算力，使用先进的通信技术实现高速可靠的数据传输，让工厂管理脱离现场；同时依靠云计算平台对海量数据进行高效处理，获取更多有价值的信息。

1.2国内外发展现状

近年来关于工业物联网系统的设计研究有着很大的进展，但产品多是以定制化的形式聚焦于高端产能用户。一些大型的制造业企业，通过自主研发的方式为自己的工厂设计了配套的物联网软硬件系统，如美的公司的MeiCloud系统[1]等，但这些系统都是高度定制且不完全开放，很难适用于形态多样的小微制造企业。对于小微制造企业来说，沿着这类模式，依靠自身能力整合软硬件资源，来设计并搭建自己的物联网系统十分困难。

针对没有物联网系统自主研发能力的企业的定制需求，一些云计算业务服务商如华为云、阿里云推出了有关工业物联网的产品，内容涉及物联网数据储存、在线数据分析、物联网应用开发等领域[2]。用户在获取第一手原始数据后，可以通过云服务商提供的配套软件工具上传至云平台，并在云端完成数据转储和分析等业务。但这类云服务产品在数据采集末端没有提供与之配套的数据采集设备。企业在使用这一类产品前，需要购买具有联网和编程能力的生产设备或数据采集终端，并自行搭建厂内通信网络。对于尚未实现生产设备数字化的中小工业企业来说，这类解决方案的门槛较高，要实现以上要求较为困难。

也有一些科技企业推出了包含软件系统和硬件设备定制化工业物联网产品，如卡奥斯(Cosmoplat)公司的高端智能装备解决方案[3]，提供包括智能工厂规划咨询、智能装备设计与制造、系统集成、产线运维等智能化服务。这类定制化施工方案包括了硬件和软件系统的建设，但对于中小微企业来说这类解决方案过于昂贵，不适合生产规模较小的使用场景。

思科（Cisco），研华（Advantech）等硬件制造商提供了多种适用于工业物联网的硬件设备，如交换机、网关[4]等，并提供了有关硬件设备接入和管理的软件产品[5]。已经实现生产数字化管理的大中型工业企业可以很方便的通过这类硬件产品实现物联网的组建，但对于尚未实现数字化管理的小微工业企业，这类硬件无法与未实现数字化控制的生产设备连接。这些企业若想享受大数据生产管理所带来的红利，则必须先对现有的生产设备进行一定程度的数字化改造以达到与标准物联网设备进行通信的基本硬件要求。如果仅仅是为了实现厂内设备的联网管理，对整个生产模式和设备都进行修改和更换，这对于升级成本有限的小微企业来说是难以接受的。

综上所述，目前全球关于工业物联网系统的产业布局尚处于一种分散的局面，各厂商间的产品缺乏互相适配的标准，没有形成一种成熟的生态。反观发展较早的机电自动化产业，已经形成了完整的解决方案模式，各大厂家间的产品实现了一定程度的通用。无论是核心设备的研发厂商还是基层的运维、安装服务提供商，都能以一种标准化的工作方式参与其中。未来工业物联网产业终将形成类似的标准化生态，各家厂商分工合作，提供可兼容、功能全面、适用面广的工业物联网软硬件产品。工业物联网从起步到成熟必定需要一个发展周期，21世纪20年代将是工业物联网产业的成型期，我们希望能弥补现有产品的不足，在这一关键时期为广大中小工业企业提供一个实用性强、成本低、的工业物联网解决方案。

1.3产品特点和创新

结合上文分析，工业物联网产业尚处于发展初级阶段，无论是物联网硬件制造商、物联网应用供应商，还是云计算服务商，都难以提供一种低成本的囊括端设备数据采集、设备间通信、云端接入、管理软件和数据应用软件的工业物联网解决方案。对于多数小微工业企业来说，整合各家的产品并搭建自己的工业物联网系统十分困难，需要专门聘请相关的技术人员进行设计。面向众多小微工业企业的需求，我们需要开发一种低成本的、通用化的工业物联网产品：用户在使用过程中无需考虑设备通信、数据处理等专业化问题，企业依靠自身的技术力量就能完成安装部署，以极低的前期投入成本享受到完整全面的智能化大数据管理体验。

实现上述目标的一个关键思路是全栈化。全栈（full stack）概念来源于软件工程学，意味着开发者或产品拥有系统技术栈的各个层次上理解问题和解决问题的能力，或者说实现和管理整个技术栈的能力。一个全栈化的物联网系统需体现出完整性：内容包括终端采集硬件、通信设备、管理平台、可视化界面等，一次部署即可实现一个完整物联网系统的全部能力。对于用户来说，设备通信、数据处理等中间环节被封装在系统内部，在部署和施工过程中无需直接接触。反观目前市场上的工业物联网产品，用户需自行购买软硬件组件并进行配置，不同供应商产品间的有着不同的通信方案，难以协同配合。而本系统则有着产品完整性的优势，复杂度低、易于施工，极大的降低了物联网系统使用难度和部署成本。

本工业物联网系统亦使用了一种创新性的数据采集模式。工业物联网产品难以在小规模工厂推广的一个主要原因是其对原有设备自动化能力要求过高，而小型工厂往往都以人工生产为主，企业也不愿意投入大量资金，冒着成本增加和可靠性降低的风险来升级自动化生产设备。对此，我们希望使用一个易适配的、通用化的数据采集硬件来对现有工厂设备施行无损升级。数据采集硬件在安装接入时，绕过了工业设备控制器，与工业设备的传感器直接相连，无需设备本身拥有数控能力。在不改变原设备人工控制能力的前提下，接入设备现有的传感器，实现设备数据联网。这种方案帮助了企业实现生产数据数字化、无纸化监控，同时也不破坏原有的传统生产模式。

1.4工作模式

面向工业设备的物联网系统需要具有三个基本功能需求：1、生产设备与局域网或互联网间的数据传输；2、对设备数据的管理和储存；3、面向设备管理者的操作界面。

分别涉及前期数据获取和采集，中段数据分发与解析，以及末端的数据可视化，其中包含多种软件和硬件设备。根据属性和功能的不同，可以把整个系统拆分为三个模块：数据采集终端、联接管理系统、客户端应用。

数据采集终端是一个安装在设备侧的数据采集和传输装置，它的作用是协助传统工业设备拥有获取自身参数的能力，实现设备信息的数字化读取。数据采集终端需要在电控水平较低的工厂部署，没有现成的设备通信接口使用，故其配备了一系列供传感器接入的模拟量接口：如获取设备的启停和荷载信息，可以通过检测设备工作电流实现；获取容器内的介质或环境信息、可以通过各式温度、压力、浓度传感器实现；机械加工中手动机床的加工轨迹、吊车的运行轨迹可以通过光栅或接触开关来获取等。这些信息经过相应的传感器转换为模拟电信号，数据采集终端需要收集各种形式的模拟信号，并在内部进行模数转换后进行储存。在通信方面，数据采集终端使用最新的窄带物联网（NB-IoT）[6]通信技术接入互联网，对比传统的有线数据采集模块，NB-IoT数据终端无需在厂内布设网线和交换机；对比使用WiFi技术的数据采集模块，NB-IoT数据终端无需安装无线路由器[7]。新型的通信技术大大简化了设备部署难度，在部署设备的过程中无需考虑网络布设及其他通信问题，仅需与各传感器电缆铰接，拥有基本电工知识的工人即可独自完成数据采集终端的安装部署。

联接管理系统是一个运行于远程服务器的软件集合，包括管理用于终端硬件接入的网络端口，处理外部应用程序的连接请求的接入控制程序，以及数据转储和分析程序等。在接入端口获取到某个终端设备上报的信息后，联接管理系统内的算法会首先对设备进行鉴权，随后自动对合法信息进行通信协议的转换，并根据设备的不同类型将设备信息分流至不同的储存空间进行储存。联接管理系统的数据分析进程会定时读取缓存内新添加的数据，获取有关设备工作状态和运行情况的多种信息。此外，联接管理系统也提供了可访问公共数据空间的API用于同客户端应用程序进行数据交换，外部应用程序可以通过API读取所有在线的设备信息和各种数据分析结果。

客户端应用是一个网页程序，通过上文所述的通信接口从联接管理系统中读取所需数据，用户可以在联网条件下通过PC和移动端设备的浏览器打开客户端应用，查看当前在线的采集终端信息，并查看各种生产参数曲线等。

这一软硬件兼备的工业物联网系统实现了最基本的工业设备数据获取、处理和呈现的全过程。在此框架上可以进行更深一步的开发，例如同设备制造厂商合作，将数据采集终端的功能集成在设备内部，或者添加更多定制化的数据分析算法进入联接管理系统。也可以开发更多形式的客户端应用，实现更高级的数据可视化效果等等。

2. 数据采集终端设计

2.1 全局设计

数据采集终端用于采集生产过程中产生的各种数据，轻便可靠，易于大批量安装。设备实际上是一种无线联网的数据采集模块，可以收集多种类型的传感器信号，同时支持人工录入数据。该设备可以以无线方式接入互联网，也可使用串行接口接入厂区现有的有线网络。

2.1.1设计指标

数据采集终端的主要设计指标如下表所示：

表1 数据采集终端的主要设计指标

项目	设计要求
体积限制	PCB面积<0.01m2，外壳< 0.2m×0.2m×0.1m
设备接入方式	电流、电压、串行接口
电流模拟量输入	量程4-20mA， 分辨率0.01mA，8通道
电压模拟量输入	量程0-1V，分辨率0.01V，8通道
有线通信	RS232，RS485（Modbus）
无线通信	窄带物联网（NB-IoT）
安全性	IP67级防护，防雷击电路，防输入过载电路
人机交互	0.96寸OLED显示屏

项目设计要求

体积限制 PCB面积<0.01m2，外壳< 0.2m×0.2m×0.1m

设备接入方式电流、电压、串行接口

电流模拟量输入量程4-20mA， 分辨率0.01mA，8通道

电压模拟量输入量程0-1V，分辨率0.01V，8通道

有线通信 RS232，RS485（Modbus）

无线通信窄带物联网（NB-IoT）

安全性 IP67级防护，防雷击电路，防输入过载电路

人机交互 0.96寸OLED显示屏

数据采集终端共提供了三种设备接入方式，可以供各种输出模拟量信号的工业传感器以及带有RS485串口的工控设备接入。电流接入模式主要面向输出电流信号的工业传感器，如压力传感器、交流电变送器、电磁流量计等，电压接入模式主要面向输出电压信号或开关量信号的工业传感器，如热点偶、光电开关等，为满足一些较为复杂仪器仪表和可编程逻辑控制器（PLC）的接入需求，数据采集终端亦提供了RS485端口，可实现采集终端与待测设备间的一对一或一对多半双工通信。

在上行通信方面，数据采集终端提供了两种通信方式，分别为串行有线通信和窄带物联网（NB-IoT）无线通信。串行接口支持RS232或RS485两种接口标准，主要用于调试和固件更新，NB-IoT无线模块负责实现数据采集终端和云服务器间的数据透传。在人机交互方面，硬件终端提供了一个0.96寸OLED显示屏用于显示设备工作状态和通信参数，此外还配备了三个输入按钮，分别用于设备复位，通信测试和菜单选择，给予了使用者极大的便利。

在安全性方面，数据采集终端配备了防雷击电路和防输入过载电路，可抵御因接地电势剧烈变化导致的电网浪涌冲击，和传感器故障产生的过载电信号，充分考虑了恶劣复杂的工业使用环境，保证终端始终可以可靠且稳定的工作。

2.1.2组成结构

数据采集终端基于嵌入式技术设计，使用低功耗的STM32L4系列微处理器作为控制单元，使用独立的模数转换（AD）采样芯片，并配备了无线通信模组和OLED显示屏。数据采集终端采用模块化的形式设计，模块与模块间使用串行接口通信，各模块间的组成模式如图1所示：

图1 数据采集终端模块组成图

其中，模数转换（ADC）芯片将接收到的电压和电流模拟信号转换为数字信号，并通过串行数据接口使用SPI协议同MCU进行

通信。ADC芯片的型号为德州仪器ADS8332，这是一款16位，8通道的ADC芯片，最高采样率为500k/s，满足设计精度要求同时具有较低的成本。无线模块使用移远通信BC35G型模组，这是一款面向物联网应用设计的NB-IoT通信模组，体积小，功能丰富。关于具体的设备选型规范和各模块的详细设计方案将在下文进行讨论。

2.2 STM32L431主控模块

主控模块由中央处理器和外围接口电路组成，负责协调各个子模块的工作，收集各个子模块上传的信息和数据，并在内部对其进行整合和处理。主控模块需要保证稳定且不间断工作，对中央处理器和外围元器件的综合性能有着一定的要求。数据采集终端选用STM32L431RCT型MCU作为主控模块的中央处理器，以满足高计算性能、高稳定性以及低功耗的设计要求。STM32系列微处理器是目前世界上最为流行的ARM架构微处理器，由意法半导体公司研发，专门面向高性能、低成本、低功耗的嵌入式应用场景[8]。STM32L431系列属于基本型超低功耗STM32处理器，在满足设计要求的前提下在功耗、性能以及成本之间作了很好的平衡。表2为该芯片的基本属性。

表2 STM32L431RCT低功耗微处理器性能参数表

参数	值	参数	值
型号	STM32L431RCT	内核型号	Cortex-M4
封装形式	LQFP	位数	32位
引脚数	64	内存大小	256KB
尺寸	12mil×12mil×1.6mil	通信接口数量	16

参数值参数值
型号 STM32L431RCT 内核型号 Cortex-M4
封装形式 LQFP 位数 32位
引脚数 64 内存大小 256KB
尺寸 12mil×12mil×1.6mil 通信接口数量 16
数据来源：STM32L431xx Datasheet - production data意法半导体

2.2.1初始化配置

STM32系列微处理器的初始化配置主要是针对芯片的硬件属性，包括芯片引脚配置、时钟配置、外设接口配置等。这些初始化参数以C代码的形式保存，在编译时与功能代码一同编译。开发者可以直接编写C代码配置文件，但涉及到的文件和代码修改量是巨大的，且需要对MCU硬件结构和原理有较为深入的理解。开发者也可以通过一些集成化的初始化生成软件来实现STM32的初始化，如ST公司官方推出的STM32Cube MX代码生成软件。

STM32CubeMX是ST公司针对STM32系列微处理器开发的一款初始化代码生成工具，它集成了多个软件平台，包括STM32Cube HAL集成库以及TCP/IP,USB,RTOS等通信中间件，可以支持STM32全系列芯片的开发[9]。STM32CubeMX拥有一个图形化的操作界面用于配置MCU的初始参数，并自动生成其所对应的初始化C代码及工程文件，开发者可以在生成的工程目录下调用其生成的库函数直接进行二次开发，而不用关心这些函数具体在底层如何实现。

需要配置的初始化项目根据功能可以分为几类，分别是针对MCU工作形态的全局配置，外设驱动的配置以及通信接口的配置。以下是针对本产品的MCU初始化配置参数说明。

2.2.1.1芯片引脚配置

STM32L431RCT共有64个引脚，除去11个电源引脚和1个RESET引脚之外，其他引脚均带有多种工作模式，可作为通信引脚、输入输出引脚模数转换引脚使用，开发者在编写程序前，需要预先配置这些引脚的功能，这一过程被称作引脚的初始化配置。

9	201614098_郭昭利_16电子信息工程4班_智能电子血压计设计 郭昭利 - 《大学生论文联合比对库》- 2020-05-14	1.2%（116） 是否引证：否
10	基于51单片机的UART串口通信 - 通信设计应用 - 《网络（http://www.elecfans.）》- 2017	1.1%（109） 是否引证：否
11	AOA定位系统基站端设计 唐豪杰 - 《大学生论文联合比对库》- 2019-05-06	1.0%（99） 是否引证：否
12	支持Avalon总线的UART串口通信模块的 设计与实现 陈哲正 - 《大学生论文联合比对库》- 2019-05-25	1.0%（99） 是否引证：否
13	支持Avalon总线的UART串口通信模块的设计与实现 陈哲正 - 《大学生论文联合比对库》- 2019-05-30	0.8%（76） 是否引证：否
14	基于实时操作系统的四旋翼飞行器设计 莫梓华 - 《大学生论文联合比对库》- 2017-05-26	0.8%（76） 是否引证：否
15	便携电阻测试仪设计 孙伟达 - 《大学生论文联合比对库》- 2018-06-02	0.8%（76） 是否引证：否
16	基于FPGA的任意波形发生器 黄西广 - 《大学生论文联合比对库》- 2019-05-12	0.8%（76） 是否引证：否
17	1507200350-黄西广-电子信息工程-基于FPGA的任意波形发生器-石松 黄西广 - 《大学生论文联合比对库》- 2019-05-14	0.8%（76） 是否引证：否
18	201614098_郭昭利_16电子信息工程4班_智能电子血压计设计 郭昭利 - 《大学生论文联合比对库》- 2020-05-15	0.8%（76） 是否引证：否
19	一种采用UART接口对智能电表MCU自动频率修调方法 闫书芳; - 《集成电路应用》- 2020-01-20 1	0.7%（72） 是否引证：否
20	基于造纸污染气体自动全分析仪的光催化降解过程中环境温/湿度影响的研究 马军(导师：沈文浩;汪华杰) - 《华南理工大学硕士论文》- 2018-04-12	0.7%（72） 是否引证：否
21	吕妍_便携式多导睡眠监测分析系统的研究与设计 吕妍 - 《大学生论文联合比对库》- 2017-05-06	0.7%（72） 是否引证：否
22	基于无线传感器网络的智能灌溉系统——数据采集系统设计 李建毅 - 《大学生论文联合比对库》- 2018-06-08	0.7%（72） 是否引证：否
23	基于单片机的汽车胎压监测器 常彬涵 - 《大学生论文联合比对库》- 2018-05-28	0.7%（72） 是否引证：否
24	2013131303-海博-郑蕊蕊 海博 - 《大学生论文联合比对库》- 2017-06-12	0.7%（71） 是否引证：否
25	基于压力传感器的电子秤设计 马明轩 - 《大学生论文联合比对库》- 2018-05-19	0.7%（71） 是否引证：否
26	基于STC12C5A60S2简易智能家居控制系统设计 陈辉煌;傅仙发; - 《电子技术》- 2016-05-25	0.7%（68） 是否引证：否
27	基于LoRa的低功耗广域物联网系统设计研究 王一波(导师：黄磊) - 《深圳大学硕士论文》- 2019-06-30	0.7%（68） 是否引证：否
28	基于UART的多机通信系统的设计 辛惠娟;刘兴智;李国荣; - 《航空维修与工程》- 2015-12-20	0.5%（51） 是否引证：否
29	二次雷达信号发生器设计 杜宗 - 《大学生论文联合比对库》- 2017-06-06	0.5%（48） 是否引证：否
30	一种改进的DCS通信接口方案及应用 蔡钧;段文伟;付俊杰;李平康; - 《自动化仪表》- 2011-08-20	0.3%（31） 是否引证：否

原文内容

，STM32CubeMX的引脚配置界面如图2所示。

图2 STM32CubeMX引脚配置界面

芯片引脚的初始化过程主要包括分配引脚类型和配置工作参数。引脚类型包括：通信引脚、输入输出引脚、通信引脚、ADC引脚和其他功能性引脚等。不同类型的引脚存在工作电平、工作频率以及电气特性的差别，必须妥善的进行初始化设定。

通信引脚包括UART（通用异步收发传输器）引脚和SPI（串行外设接口），这两者均属于串行通信协议。UART是一种通用串行数据总线，用于异步通信。该总线双向通信，可以实现全双工传输和接收[10]。在嵌入式设计中，UART用于主机与辅助设备通信。此处我们设置了两组UART引脚，分别作用于电脑和采集模块、采集模块和Nb-IoT之间的通信。通信及其他功能引脚的配置参数如表3所示

名称	标识	类型	功能

PA9	USART1_TX	UART	外部通信
PA10	USART1_RX		
PC4	USART3_TX	UART	NB-IoT模组通信
PC5	USART3_RX		
PB1	KEY1	GPIO	按钮1
PB3	KEY2		按钮2
PA1	SPI1_SCK	SPI	ADC芯片通信
PA6	SPI1_MISO		
PA7	SPI1_MOSI		
PB7	GPIO_PIN_7	GPIO	OLED屏幕虚拟I2C接口
PB6	GPIO_PIN_6		
PC13	LED		LED指示灯

名称标识类型功能

PA9 USART1_TX UART 外部通信

PA10 USART1_RX

PC4 USART3_TX UART NB-IoT模组通信

PC5 USART3_RX

PB1 KEY1 GPIO 按钮1

PB3 KEY2 按钮2

PA1 SPI1_SCK SPI

ADC芯片通信

PA6 SPI1_MISO

PA7 SPI1_MOSI

PB7 GPIO_PIN_7 GPIO OLED屏幕虚拟I2C接口

PB6 GPIO_PIN_6

PC13 LED LED指示灯

表3 主控芯片引脚配置表

2.2.1.2芯片时钟配置

时钟是微处理器的必备部件之一，为微处理器提供标准的工作频率，推动微处理器内各个部分执行相应的指令。STM32L431RCT采用多时钟源设计，目的是为不同工作频率的外设提供不同的时钟信号，以达到性能和功耗的最优化。这些时钟源最初都由同一个系统时钟信号分频得来，开发人员可以通过修改配置文件来定义芯片内部的分频机制，以产生不同种类外设所需的不同频率的时钟信号。

可以通过STM32CubeMX软件的图形化时钟配置界面来快速、直观的配置芯片时钟。图3是STM32CubeMX中显示的STM32L431RCT型芯片的时钟系统图：

图3 STM32L431RCT时钟系统图

图中左上部分显示了STM32的四个独立时钟源，分别为HIS（高速内部时钟），HSE（高速外部时钟），LSI（低速内部时钟），LSE（低速外部时钟），其中高速时钟经过分频或倍频处理可作为系统时钟使用。此处配置HSE（高速外部时钟）作为主系统时钟，外部时钟信号由独立的晶振提供，其精度和稳定性优于微处理器自带的内部时钟晶振。在使能HSE（高速外部时钟）后，来自HSE的晶振脉冲经过AHB分频器后变为主时钟HCLK信号，随后被复用为多路，其中一路时钟信号PCLK1将被用于UART1，UART3和SPI等外设。

为节省系统功耗，设计时选用较低频率的8MHz晶振作为时钟源，不经过分频和倍频即可提供给主时钟HCLK和外设时钟PCLK1使用。图中右侧可见各外设的工作频率，全部为8MHz，与主时钟保持一致。

2.2.2 内部通信参数

数据采集终端内部各模块使用串行总线进行通信，MUC通过串行总线接收无线通信模块或AD模块发会的数据，也通过串行总线驱动OLED屏幕显示字符内容。数据采集终端共使用了三种不同的串行总线，分别为通用异步收发器（UART）、串行外设接口（SPI）和I2C总线。下面将对这三种不同总线的功能及配置情况做详细说明。

UART全称为通用异步收发传输器（Universal Asynchronous Receiver、Transmitter），俗称“串口“，是一个完成串并转换的硬件，也同时负责数据格式的编码和解析，具有硬件和协议的双重属性[10]。USART是UART的低功耗版本，在数据采集终端中共使用了两个USART，UASRT1是数据采集终端的主串口，用于同外部设备通信，如工控主机、PLC等。 USART3用于MCU与NB模块间的通信，传输不同的AT指令。UART的通信规则十分简单，一个数据包即是一个字节流，内部都是有效通信数据，没有定义任何标记位或校验位。由于接收到的报文长度未知，且报文不一定带有结束符，所以必须使用一定机制来判断一段数据的接收是否结束。一种做法是使用超时接收：当串口超过一定时间未接收到数据，即可判断数据接收完毕。STM32在硬件上为此类算法做了专门的优化，在检测到UART总线空闲时，状态寄存器（USART_SR）中的IDLE（串口空闲中断）位会被置位，并产生一个中断，告知系统一个完整的串口数据帧已被接收。通过在中断程序内编写算法即可实现一个完整串口数据包的读取，其优点十分明显：接收数据时产生的中断不会与主程序产生冲突，不会对运行速度造成较大的影响[11]。表4是两个UART串口的通信配置参数：

	波特率	字长	校验位	停止位
USART1	115200	8bit	0	1
USART3	9600	8bit	0	1
	全局中断	模式	流控	中断优先级
UASRT1	使能	异步	无	1
USART3	使能	异步	无	2

波特率字长校验位停止位
USART1 115200 8bit 0 1
USART3 9600 8bit 0 1
全局中断模式流控中断优先级
UASRT1 使能异步无 1
USART3 使能异步无 2
表4 USART1与USART3通信参数

I2C总线由两条信号线组成，其中一条用于传输数据（SDA），另一条用于传输时钟信号（SCL），它是一种同步通信协议，即通信的AB双方以时钟线（SCL）连接，接收方从机工作在发送方主机传输的时钟信号下[12]。MCU使用I2C协议与OLED屏幕进行通信。关于I2C协议的详细内容不在本文讨论范围内，在此不做过多介绍。

SPI总线用于MCU与ADC芯片ADS8332间的数据通信。SPI是Motorola公司推出的一种同步串行接口技术，是一种高速的，全双工，同步的通信总线。共有四条线路，分别为SS（Slave Select 从设备使能），SCK（Serial Clock串行时钟），MOSI（Master Output Slave Input主设备输出/从设备输入）,MISO(Master Input Slave Output主设备输入/从设备输出)[13]。关于SPI协议的详细内容不在本文讨论范围内，在此不做过多介绍。

2.2.3 外围电路设计

芯片外围电路需要为芯片提供稳定的电压源与振动源以保证芯片的基本工作需求。图4是STM32L431RCT主控芯片的外围电路原理图，图中可见起振电路、按钮控制电路和芯片接口配置情况。STM32L431的工作电平为3.3v，共有6个电源输入引脚。由于引脚在电平翻转时会产生尖峰电流，为获得良好的工作效果，每组电源引脚都配备了0.1uF的去耦电容来减小尖峰电流对旁路器件的影响。此外，为缓解按键在复杂使用环境中可能出现的抖动效应，每组按键都并联了大小为10nF电容以消除抖动效应。

图4 STM32L431RCT芯片外围电路原理图

图5是整个数据采集模块的电源电路原理图。电源额定输入电压12V，输出电压3.3V，图5中电路由左至右分别为输入保护电路、电压隔离电路和降压稳压电路。输入保护电路具有防过载、防反接、防静电等保护功能，由 1N4148二极管，0.2A自恢复保险丝与SMBJ-15CA瞬态电压抑制二极管（TVS）组成。1N4148与自恢复保险丝串联构成了一个基本的防反接和防过载电路，随后在输出末端并联反向击穿电压为15V的TVS二极管，在极端情况下可以瞬间反向导通，保证输出电压被钳位在安全范围内。为了实现更好的隔离效果，在保护电路末端添加了额定功率为1W的DC/DC模块电源HLK-1D1205，可以实现输入端与输出端之间的电压隔离，进一步提升抗干扰性能。由于HLK-1D1205的输出电压较高且精度存在较大偏差（5V±15%），在模块电源输出末级又添加了三端稳压集成电路AMS1117-3.3，最终输出稳定的3.3V电压。

图5 电源电路原理图

2.3 通信模块设计

2.3.1 NB-IoT无线通信

2.3.1.1 NB-IoT技术简介

NB-IoT（Narrow Band IoT）是一种构建于蜂窝网络中的新兴通信技术，在2017年实现商用，主要用于低功耗广覆盖的物联网应用中[6]。NB-IoT的特点是覆盖范围广，功耗低，稳定性高，成本低，支持LTE制式平滑演进，可以根据不同运营商的需求灵活调整频段和部署模式。

NB-IoT的物理层实现与LTE类似，但更加精简。其上行技术使用SC-FDMA（单载波频分多址），下行技术为OFDMA，系统带宽为180kHZ，可以布置于LTE系统的保护带或边缘无用的频带，在手机信号覆盖范围内都可以正常通信。此外，NB-IoT专门针对物联网的严苛应用场景进行了算法优化，添加了重传机制，加上较窄的带宽，相比GSM有20dB+增益[14]，拥有极强的穿透性且可以覆盖更广的地域。

NB-IoT主要依靠动态休眠的通信模式来实现超低功耗。不同于手机需要随时侦听网络中可能发起的请求，NB-IoT模组在发送数据包后，会立刻进入休眠状态，直到下一次有上报数据请求的时刻再开始新一轮的通信活动。由于物联网数据采集的行为习惯，多数时间模组都会处于休眠状态，此时的功耗将非常低。此外3GPP组织还定义了NB-IoT的空闲态eDRX（扩展非连续接收）功能，将寻呼周期从传统的2.56秒拓展到2.92小时，减少寻呼行为所造成的功耗[15]。

对比其他可能的物联网通信方案，NB-IoT有非常显著的优势。相比有线通信方案，无线通信方案极大的降低了组网成本，且可以灵活布置终端机的位置，新终端机也可以不受基础硬件的限制快速的部署上线。在无线通信方案中，NB-IoT基于现有的蜂窝移动网络，覆盖范围广，网络质量好。对比WiFi方案，NB-IoT不需要在厂区内部布设有线网络和无线路由器；对比传统的无线电通信方案，NB-IoT不需要配置高功率的发射机也无需申请合法的通信频道许可；对比LoRa（Long Range Radio远距离无线电）方案[16]，NB-IoT无需在厂区内部自行搭建LoRa网络，也不需要配置专门的网关设备。总而言之，NB-IoT是一种十分合适的工业物联网系统通信手段。

2.3.1.2 模组驱动电路设计

数据采集终端使用移远通信BC-35G模块接入NB-IoT网络。BC-35G是移远通信（QUECTEL）于2018年发布的一款单频段LTE Cat NB2 模块，使用海思Boudica 150芯片。此款模块在移远通信BC系列NB-IoT模块中属于中端型号，支持450-470MHz，696-960MHz，1695-2180MHz等多个通信频段，且支持CoAP/LwM2M/TCP/MQTT等多种物联网常见协议。BC-35G使用了华为海思的基带芯片，已通过华为物联网平台接入认证，相比其他厂商生产的NB-IoT模块，BC-35G可以直接使用内置AT指令接入华为物联网平台，无需复杂的接入认证程序。

为保障BC-35G模块的正常工作，需要设计相应的外围电路，如供电电路、串口电路、天线电路等，此外由于BC-35G使用外置USIM接口，设计者还需要单独设计外置USIM卡电路，以保障模块正常入网。由于BC-35G模块高度集成化，体积较小，耐压和抗静电能力较弱，外围驱动电路需要优异的抗干扰性能以保证其在工业环境中不会因电气干扰而损坏。

图6为BC35G模块外围电路原理图，图内左上角为BC35G模块，右上角为SRV04型TVS二极管阵列以及USIM卡插槽。SRV04是专为USIM卡保护而设计的TVS二极管阵列，内共有9个TVS二极管，其反向击穿电压低于USIM卡最高输入电压，可以实现对USIM卡良好的静电和浪涌保护。图内左下角为串口电压转换电路，由于BC35G模块串口输出电平（1.2V）低于MCU电平（3.3V），需进行电平转换。此处使用8550型三极管实现串口信号的降压和升压。

图6 BC35G模块外围电路原理图

2.1.2 Modbus有线通信

Modbus协议是工业领域通信协议的业界标准，是工业电子设备之间最常用的连接方式之一。Modbus支持多种传输介质，最常用的是使用以太网进行传输的Modbus TCP以及使用RS485总线进行传输的Modbus RTU[17]。在Modbus是一种主从形式的通信协议，在一条总线上配备有一个主机（Master）和多个从机（Slave），在实际应用中，主机对应区域内的核心工控电脑，从机对应区域内各设备的控制单元。图7描述了一个拥有1个主机和4个从机的Modbus通信网络。

图7 Modbus通信网络示意图

Modbus协议是一种十分轻量化的通信协议，以使用RS485标准传输的Modbus RTU协议为例，其通讯帧格式如图8所示：

图8 Modbus协议格式示意图

其中，地址码为从机的地址标识，在一个通信网络中是唯一的，由设计者决定；功能码是主机所发的功能指示，如读写寄存器等；数据区是通信具体数据；CRC码是使用CRC算法生成的两字节长度的校验码。数据采集终端支持使用Modbus协议读取各采样通道数据及设置各种内部参数，针对该设备的详细Modbus通信规则将在2.5.3节介绍。

2.4 传感器接入电路

数据采集模块有8路传感器接口，用户可以通过通道开关选择电压接入或电流接入模式，其中电压信号范围为0-1V，电流信号范围为4-20mA。输入信号经过滤波电路和过载保护电路后被输送至模数转换模块转换为数字信号，随后发送给MCU。

数据采集模块使用了独立的模数转换芯片ADS8332实现高精度的模拟信号采样，其性能参数如表5所示。ADS8332是一款16位采样芯片，意味着当参考信号为1V时，其采样分辨率最高可达0.16mV，可满足工业生产中各种严苛的信号精度要求。

表5 ADS8332性能参数表

参数	值	参数	值
型号	ADS8332IPWR	A/D位数	16
封装形式	TSSOP-24	采样率	500K
工作电压	2.7V-3.6V,1.65V-5.5V	输入类型	差分
参考信号类型	外部电压	通信接口类型	SPI

参数值参数值

型号 ADS8332IPWR A/D位数 16

封装形式 TSSOP-24 采样率 500K

工作电压 2.7V-3.6V,1.65V-5.5V 输入类型差分

参考信号类型外部电压通信接口类型 SPI

数据来源：ADS8332IPWR Datasheet - production data

ADS8332使用复用器实现多通道数据采样，因此不同通道的采样必须安排在不同时间进行，其作用效果等同于一个带有复用器的单通道ADC模块。相比于使用独立复用器+单通道ADC芯片方案，ADS8332在内部已经集成了复用器，有着芯片体积优势，可以大大节省电路板空间，简化布线设计。ADS8332的8路复用器与ADC单元相互独立，在芯片内部并没有直接输出至ADC模块，而是输出至外部引脚MUXOUT。设计者需添加MUXOUT至采样输入引脚ADCIN间的转换电路。为了限制ADCIN的输入电流，提高输入阻抗，我们在复用器输出MUXOUT与采样输入ADCIN间添加了运算放大器OPA320，实现输入的缓冲功能，同时OPA320带有最高2000V的ESD保护，一定程度上可以防止浪涌信号损害ADC芯片。

图9为传感器接入电路的电路原理图，左上角转换后的各通道电压信号，经过一级滤波后输入至ADS8332的8路复用器。右上角为输入信号隔离电路，其本质是一个基于运算放大器OPA320实现的信号跟随电路。原理图左下方为2.5V精密基准电压源电路，由LTC6655精密电压源芯片提供准确的2.5V基准电压，作为ADS8332的采样参考。

图9 传感器接入电路原理图

图10 传感器接入电路测试板1

2.5 工作机制

2.5.1模拟信号采样

MCU与ADC芯片通过SPI总线通信，在采样开始时，MCU向ADC芯片发送开始采样指令，并指定特定的读取通道，随后ADC芯片开始采样，采样结果随后通过SPI总线发送至MCU，至此一轮采样结束。各通道的采样频率可由用户自定义设置，但需要与NB-IoT的数据上报周期保持一致。

MCU通过修改ADS8332中的指令寄存器（Command Register CMR）和配置寄存器（Configuration Register CFR）来控制ADS8332的工作模式和工作行为，其中CMR寄存器用于通道选择、触发采样和数据读取，CFR寄存器用于设置采样触发时钟源和触发模式等。表6为数据采集终端MCU需要使用的ADC芯片CMR寄存器控制指令真值表。

表5 ADS8332 指令寄存器CMR主要指令真值表

D15	D14	D13	D12	指令
0	0	0	0	选择输入通道0
0 0 00000011	0 0 0 1 1 1 1 0 1	0 1 1 0 0 1 1 1 0	1 0 1 0 1 0 1 1 1	选择输入通道1 选择输入通道2 选择输入通道3 选择输入通道4 选择输入通道5 选择输入通道6 选择输入通道7 唤醒ADC模块 读取采样值

D15 D14 D13 D12 指令

0 0 0 0 选择输入通道0

0

0

0

0

0

0
0
1
1 0
0
0
1
1
1
1
0
1 0
1
1
0
0
1
1
1
0 1
0
1
0
1
0
1
1
1 选择输入通道1
选择输入通道2
选择输入通道3
选择输入通道4
选择输入通道5
选择输入通道6
选择输入通道7
唤醒ADC模块
读取采样值
数据来源：ADS8332IPWR Datasheet - Command Set Defined by Command Register(CMR)

采样周期由MCU内部定时器TIM1控制，TIM1经多级分频后每一秒钟触发一次定时器中断，每次中断都会使一个分频计数器的值递增，当达到分频计数器达到预设分频周期时，MCU执行触发采样程序，同时分频计数器清零。用户可以通过Modbus指令修改分频周期寄存器的大小来改变采样周期，例如当分频周期寄存器设置为60时，采样周期为60s。MCU中触发采样程序的原理是，先通过SPI总线修改ADS8332中CMR寄存器的高四位，选择所需要的采样通道，经过4轮ADS8332时钟周期后设置生效，随后，MCU通过SPI总线修改ADS8332中CMR寄存器的高四位为1011，等待设置生效后触发ADC芯片进入工作模式，MCU随后进入0.2s线程等待，此时ADC芯片内部进行AD转换工作，在线程等待结束后，MCU修改CMR寄存器的高四位为1101，之后ADS8332会通过SPI总线返回16位AD转换结果。

2.5.2无线通信

MCU与NB-IoT模块间使用串口USART3连接，通过AT命令来控制与NB-IoT模块间的交互行为。在数据采集模块开机后，NB-IoT模块开始初始化，这个时间约为10秒。MCU会在NB-IoT结束初始化后发送AT指令读取NB-IoT模块的内部参数，验证初始化是否成功。图11展示了数据采集终端开机后无线通信模块的初始化过程：

图11 无线通信模块初始化过程

其中IP地址119.3.250.80：5683是华为云设备接入服务CDP服务器的CoAP协议接入地址。发送“AT+CSQ”查询信号强度将返回模块从基站接收到的信号强度指示RSSI和信道误码率BER，这些参数将被显示在OLED屏幕上供用户参考。

数据采集终端使用AT指令将各通道的采样值发送至CDP服务器，其命令格式为：“AT+NMGS=<length>,<data>[,<seq_num>]”其中length字段是发送数据的长度，data字段是字符串类型的待传输数据，其格式为16进制。每个通道的采样值被编码为2字节供4位的16进制数据，最大值为ffff，共有8个通道，数据字段的总长度为16字节。编码样例参考图12。

图12 数据发送AT指令编码示意图

AT指令的发送周期由内部定时器TIM2控制，TIM2经多级分频后每一秒钟触发一次定时器中断，每次中断都会使一个分频计数器的值递增，当达到分频计数器达到预设分频周期时，系统触发AT指令发送程序，同时分频计数器清零。用户可以通过Modbus指令修改分频周期寄存器的大小来改变AT指令的发送周期，例如当分频周期寄存器设置为60时，AT指令的发送周期为60s。

2.5.3 Modbus通信算法

除NB-IoT无线通信外，使用者也可以通过Modbus有线通信的方式获取各通道的采样数据。此外，用户可以通过不同的Modbus指令来设置数据采集终端的各种内部参数。表6为数据采集终端的Modbus通信点位表。其中每个寄存器长度为4字节，可储存32位二进制数据。其中寄存器40001至40004为采样通道值，每个寄存器包含两路采样通道，寄存器40005为设备状态值，寄存器40006为采样和通信周期值。

串口USART1用于处理数据采集终端和外部设备间的Modbus通信，当USART1接收到一段字节流时会触发串口空闲中断，该中断会调用Modbus协议解析程序对接收到的字节流进行解析，并进行CRC校验。对于读取类命令，成功解析后，数据采集终端会返回相应的Modbus回复帧，若解析失败，数据采集终端OLED屏幕将短暂显示“ILLEGAL”字样。对于设置类命令，当设置成功后，数据采集终端OLED屏幕将短暂显示“SET OK”字样。

表6 数据采集终端Modbus通信点位表

点位	数据类型	值	备注
40001.00-03	16bit INT	采样通道1	
40001.04-07	16bit INT	采样通道2	
40002.00-03	16bit INT	采样通道3	
40002.04-07	16bit INT	采样通道4	
40003.00-03	16bit INT	采样通道5	
40003.04-07	16bit INT	采样通道6	
40004.00-03	16bit INT	采样通道7	
40004.04-07	16bit INT	采样通道8	
40005.00	BOOL	采样状态	00错误，01正常
40005.01	BOOL	NB入网状态	00错误，01正常
40005.02	4bit INT	信号强度	
40006.00	4bit INT	采样周期	单位（秒）
40006.01	4bit INT	数据发送周期	单位（秒）
40006.02	4bit INT	保留	

点位数据类型值备注

40001.00-03 16bit INT 采样通道1

40001.04-07 16bit INT 采样通道2

40002.00-03 16bit INT 采样通道3

40002.04-07 16bit INT 采样通道4

40003.00-03 16bit INT 采样通道5

40003.04-07 16bit INT 采样通道6

40004.00-03 16bit INT 采样通道7

40004.04-07 16bit INT 采样通道8

40005.00 BOOL 采样状态 00错误，01正常

40005.01 BOOL NB入网状态 00错误，01正常

40005.02 4bit INT 信号强度

40006.00 4bit INT 采样周期单位（秒）

40006.01 4bit INT 数据发送周期单位（秒）

40006.02 4bit INT 保留

3. 联接管理系统

联接管理系统是一个运行在云服务器上的程序系统，提供虚拟网络接口供分布在各地的数据采集终端接入，并对其上传的数据进行分析和管理。可以将联接管理系统视为一个中间件，即一种独立的系统软件或服务程序，位于终端硬件、基本网络服务和数据库之上，在应用软件之下，衔接网络上应用系统的各个部分或不同的应用[18]。联接管理系统这一中间件承担了数据终端和客户端界面间的信息转发和处理工作，为数据终端硬件提供了通信支持，也为上层应用软件提供了获取信息的接口（图13），能使物联网数据的整合和分析任务在一个相对封闭和独立的且易于实现高性能的计算空间中完成。

图13 联接管理系统功能示意图

3.1 全局设计

3.1.1系统工作模式

联接管理系统所要实现的具体功能有如下：1、与运营商网络连接，定时下载转储数据采集终端上报的信息；2、对终端数据进行数据格式和协议的转换；3、按照一定规则对数据进行拆解和分块储存，并做双重备份；4、对物联网数据进行实时和非实时的分析。 5、提供面向应用侧的API接口，支持客户端应用程序在线获取信息。每一个功能都拥有其对应的工作模式，都涉及到多个软件的协同工作。

以上功能被拆分为三个主要程序，分别为接入管理程序，数据处理程序和客户端应用接口程序，三个主程序异步运行，互不干扰。图14描述了数据包在三个主程序之间的传递过程和程序内部的工作模式。

图14 联接管理系统内部工作模式图

接入管理程序通过监听一个特定的网络端口来获取数据采集终端定时上报的数据，数据采集终端通过NB-IoT网络与互联网连接，按照预设的上报周期向联接管理系统的设备接入端口发送按照Modbus协议编码的字符串，字符串中包含有数据值，设备识别码，数据类型，采集时间等参数，在被联接管理系统捕获后，按照一定规则将数据转储至云端缓存中。

疑似剽窃文字表述

1. STM32CubeMX的引脚配置界面如图2所示。
   图2 STM32CubeMX引脚配置界面
2. 是Motorola公司推出的一种同步串行接口技术，是一种高速的，全双工，同步的通信总线。

3. 工业物联网数据管理信息系统与终端硬件设计_第3部分

总字数：9026

相似文献列表

去除本人文献复制比：3.5%(318)　　文字复制比：3.5%(318)　　疑似剽窃观点：(0)

1	过程化考核系统——试卷管理 闫冉 - 《大学生论文联合比对库》- 2014-05-29	1.5%（136） 是否引证：否
2	Web前端开发技术探讨 杨毅; - 《电脑知识与技术》- 2014-08-15	1.4%（125） 是否引证：否
3	罗超群_1120102000_基于LAMP架构的用户角色权限管理系统的设计与实现 罗超群 - 《大学生论文联合比对库》- 2014-06-05	1.3%（120） 是否引证：否
4	超市顾客应用平台 阚迪生 - 《大学生论文联合比对库》- 2013-04-26	1.3%（114） 是否引证：否
5	基于ACO的智能旅游景区路线规划系统设计 郝标;谭云兰;王伟年;贾金原; - 《井冈山大学学报(自然科学版)》- 2015-01-15	1.2%（108） 是否引证：否
6	基于移动终端的服务接入与服务定制的设计与实现 章武(导师：朱洪波) - 《南京邮电大学硕士论文》- 2015-03-01	1.2%（108） 是否引证：否
7	面向叙事的地图可视化方法研究 马越(导师：夏春林) - 《辽宁工程技术大学硕士论文》- 2015-01-01	1.2%（108） 是否引证：否
8	信息学院+0309102416+陈泽伟+原文 陈泽伟 - 《大学生论文联合比对库》- 2013-06-25	1.2%（108） 是否引证：否
9	20-2010201108-钱正元 钱正元 - 《大学生论文联合比对库》- 2014-06-26	1.2%（108） 是否引证：否
10	吴昊_201005570217_电子商务1001（林华珍，文献综述） 吴昊 - 《大学生论文联合比对库》- 2014-04-15	1.2%（108） 是否引证：否
11	程序设计语言类在线课程的构建 鲍小忠; - 《电脑知识与技术》- 2014-12-25	1.2%（107） 是否引证：否
12	基于Django框架的校园预约打印网站设计与实现 周玥(导师：初剑峰) - 《吉林大学硕士论文》- 2013-06-01	1.2%（107） 是否引证：否
13	09205213_陆云良 陆云良 - 《大学生论文联合比对库》- 2013-06-21	1.2%（107） 是否引证：否
14	95_刘羽舒_GitHub社区开发者角色的演化分析 刘羽舒 - 《高职高专院校联合比对库》- 2018-05-23	1.2%（105） 是否引证：否
15	080902_ 20102110010235_杨勇_LW 杨勇 - 《大学生论文联合比对库》- 2014-06-13	1.2%（105） 是否引证：否
16	基于物联网的环境监控平台设计与实现 陈也(导师：罗红) - 《北京邮电大学硕士论文》- 2014-03-07	1.2%（104） 是否引证：否
17	基于Web GIS的房产管理信息系统研究 刘惠敏(导师：宋健民) - 《郑州大学硕士论文》- 2015-05-01	1.1%（95） 是否引证：否
18	突发疫情远程智能防控诊断平台设计的研究 高长婴;袁太兴;崔晶蕾;李洪军;王辉;王中山;刘景鑫; - 《中国医疗设备》- 2020-06-10	1.0%（91） 是否引证：否
19	基于ThinkPHP框架的微课教学系统设计与实现 杜定强; - 《电脑编程技巧与维护》- 2016-06-03	1.0%（91） 是否引证：否
20	基于QoS信息的服务推荐 李智(导师：曹健) - 《上海交通大学硕士论文》- 2015-01-15	1.0%（91） 是否引证：否
21	中小家电物料库存系统分析与设计 顾小志 - 《高职高专院校联合比对库》- 2019-05-21	1.0%（90） 是否引证：否
22	高校计算机实验上机预约管理系统的设计和实现 赵争 - 《大学生论文联合比对库》- 2014-05-29	1.0%（90） 是否引证：否

23	异构数据联合检索系统的设计与实现	0.8%（73）
	高巍(导师：陈东明;李铁) - 《东北大学硕士论文》- 2013-05-01	是否引证：否
24	基于React的招聘资讯APP	0.8%（68）
	张俊峰 - 《大学生论文联合比对库》- 2019-05-16	是否引证：否
25	基于翻转课堂教学的《多媒体课件设计与制作》课程的开发研究与实践	0.6%（54）
	龚彦(导师：郑小军) - 《广西师范学院硕士论文》- 2015-06-01	是否引证：否

原文内容

使用Modbus协议进行编码的好处是它拥有较小的信息冗余，可以大幅减小数据包的大小。此外，接入管理程序在接收数据时会对Modbus字符串进行解析，并以json格式的字符串保存，使用json格式将有利于后期信息的处理，因为多数高级程序语言都有针对json格式开发的插件。

数据处理程序将定时的读取缓存中的json字符串，按照数据记录的发生时间和信息类型拆分数据包，并储存至实时数据缓存中。数据经过拆解和分类储存后，后期数据处理进程就可以直接按照类型索引在数据库中查找有关数据，而不用遍历原始的未拆解的json字符串来筛选信息，可以降低程序的复杂性并提升运行速度。此外，数据处理程序亦可以对物联网数据进行分析，通过调用一个独立的数据分析模块来实现，该模块可将json格式的数据转换为numpy类型。开发者可以修改该部分的代码来实现各种自定义的数据分析算法，经过分析处理后的结果由数据处理程序上传至云端储存器长期储存。

客户端应用接口部署于应用侧，拥有实时数据缓存和云储存的访问权限，可以根据客户端应用的需求调用所需数据资源，用于联接管理系统和客户端应用间的数据交互。

3.1.2软件组成架构

软件采用了一种扁平化、低耦合、低成本的架构设计方案。其中扁平化的特点体现在系统内各个模块都是以一种平等的方式协同工作，不存在中央控制机制。这种平等协同工作方式的实现依赖于各个模块都设置了输入和输出的缓存机制，使得各个模块可以按照不同的工作周期独立工作，而不是等待上一步完成后再被前一个进程调用。因为其弱化了进程间的相互控制和依赖，所以这种方式可以很大程度降低模块间的耦合度。这种扁平化、低耦合的架构十分适合物联网数据处理，因为各个模块独立工作，每个模块的工作机制相对简单、易于实现高可靠性，可以满足较长时间的无托管运行。这种架构也存在缺点，即整个系统响应速度受模块间缓存影响较为缓慢，但由于物联网数据更新周期长（秒级），对实时性要求不高，因此采用本架构并不会对软件综合性能产生较大的影响。关于联接管理系统的主要源代码请参考附录B。

软件组成架构如图15所示，除去数据分析算法模块与主程序存在调用关系外，其他模块间都是以缓存的形式进行异步的数据交互。在具体实现方案中，我们使用了多种商用技术，如开发接入管理程序使用的华为云IoTDA[19]平台，华为云DIS[20]和OBS[21]储存服务等，关于它们的详细介绍将在下文给出。这些开发平台框架在很大程度上减轻了开发压力，同时具备较高的性能和稳定性，且商用数据接入接口和云储存服务相比自建服务器有着明显的成本优势。

图15 联接管理系统软件架构示意图

整套系统被部署于由华为云提供的远程服务器上，其中接入管理程序在华为云IoTDA的自有服务器运行并接受管理，python主程序与数据分析算法在一台单独的云服务器上运行。根据前文所述的缓存耦合机制，所有模块间的数据交换都需要通过云储存介质，即模块输入和输出数据都是从云储存介质中提取，两个模块间无直接的数据传输通道，当其中一个模块出现故障时，其他模块仍能从云储存介质中提取数据保持自身的正常工作。如当python主程序阻塞时，接入管理程序仍能正常转储设备上报的物联网数据，保证数据不会丢失。

3.1.3开发平台与工具

联接管理系统包括多个程序组件，分别实现设备接入、数据储存、数据分析等软件功能，实现这些功能依赖于一些商用软件开发平台和工具。其中具有代表性的，如华为云对象储存服务（OBS）与数据接入服务（DIS）在时序数据的高效转储中发挥了主要作用，以下将进行简要介绍。

对象储存服务（Object Storage Service，简称OBS），是华为技术有限公司开发的一款稳定、安全、高效、易用的云储存服务，其特性是可以储存任意数量和形式的非结构化数据[21]。使用OBS服务来实现物联网数据的云储存相比于本地储存更为安全，且支持多协议访问。其安全性体现在于OBS中的特定数据由访问密钥（AK/SK）保护，无关用户在未经管理员授权的情况下无OBS的访问权限。它还提供了基于HTTP/HTTPS协议的Web服务接口，用户可以随时随地连接到Internet的电脑上，通过OBS管理控制台或各种OBS工具访问和管理存储在OBS中的数据。联接管理系统基于OBS服务实现物联网数据的长期储存，关于数据储存的机制将在3.2.3“数据储存”中详细说明。

数据接入服务（Data Ingestion Service，简称DIS），是华为技术有限公司开发的一款云服务，为处理或分析数据流数据的自定义应用程序构建数据流管道，解决云服务外的数据实时传输到云服务内的问题[20]，可用于快速构建实时数据应用。联接管理系统的一个基本功能是物联网设备的接入，通过DIS通道可将物联网设备上报的数据实时传输到云服务内，且联接管理系统的其他进程也可以实时的从DIS通道中读取数据。关于物联网设备通过DIS接入云服务的细节可参阅3.2.1“设备接入原理”。

3.2 数据转储机制

3.2.1设备接入原理

数据采集终端通过NB-IoT技术接入联接管理系统，并使用CoAP协议将业务数据上报到云端。受限制的应用协议（Constrained Application Protocol,检测CoAP）协议是一种小巧的应用层协议[22]，多被运用于小型设备或物联网设备。CoAP物联网设备使用的CoAP协议与浏览器使用的HTTP协议类似，都用于网络应用层的信息交换。可以将CoAP协议看作为一种简化的HTTP协议，HTTP是一种较为复杂的应用层协议，运行于TCP之上，对硬件的要求较高，而CoAP协议运行于UDP之上，且数据包非常小巧，适合嵌入式设备运行。 NB-IoT模组提供了一系列关于CoAP协议的AT指令，可以向特定服务器发送CoAP报文，只需在运行联接管理系统的服务器中设定一个网络端口用于接收各个NB-IoT模组上报的CoAP报文，即可接收各数据采集终端上报的业务数据。

华为云针对这种典型的应用场景开发了“IoT设备接入云服务”，提供了一个稳定的供CoAP协议接入的CDP(客户数据平台

)服务器端口，在接收到数据采集终端上报的CoAP报文后，会自动对其进行解码，并将其中的有效信息转储至DIS服务。联接管理系统的主进程只要访问DIS服务即可获得实时的业务数据。图16为设备接入过程的示意图。

图16 设备接入原理示意图

使用DIS实现设备接入无需自行搭建CDP服务器用于监听CoAP报文，也无需自行编写CoAP报文的解码程序来提取报文中的有效数据，大大减轻了开发难度。使用成熟稳定的商用CDP服务也有助于提升系统整体的可靠性。有关DIS接入的程序代码请参考附录B3。

3.2.2数据储存

物联网数据以json格式文件储存在OBS仓库中，使用文件方式储存相较于传统的关系数据库有其独特优势。由于物联网数据具有明显的时序特性，传统的关系数据库很难对这类数据实现高效的储存。如果使用关系数据库来储存物联网时序数据，数据的唯一性由其时间决定，这也就意味着数据表的主键必须设定为数据产生的时间，这样一来，数据表的行数就与记录数据的条数成正比。当数据条数随时间增长，数据表的行数会达到一个巨大的数量级，此时关系数据库的工作性能将大大降低。通过分析物联网时序数据的存储与读取特性，可以发现其储存行为是一次性的，即储存过后不会被更改；其读取行为多是根据一个确定的时间范围进行检索。使用文件可以很低成本的实现大量数据的储存，同时也可以按照数据产生的不同时间来构建文件系统，当使用时间范围检索数据时，可以很方便的定位到所需时间范围的文件，实现不亚于关系数据库的查询性能。

4. 客户端应用软件

客户端应用软件基于物联网核心应用场景设计，将后台储存的物联网数据可视化，并提供控制接口供用户和设备管理人员使用。本客户端应用软件根据典型的工业物联网应用场景设计了多种类型的设备状态监控界面，可显示实时数据和历史数据曲线，同时可在此基础上开发多种数据分析界面。软件基于WEB技术实现，在浏览器上运行，满足了PC、手机、PAD多种终端的使用需求。

4.1 全局设计

4.1.1系统工作模式

软件基于WEB技术开发，支持多平台运行，拥有独立的后端（服务器端）和前端（客户端）进程，后端进程运行于远程服务器，负责与数据库对接，并响应前端进程的请求。前端进程运行于客户浏览器端，负责在本地渲染网页效果，并将后端发回的数据布设在网站界面中。这种工作模式的好处是，后端程序运行于较高性能的远程服务器，拥有更强的计算能力和更高的网络链路速度，可以更快速的从远程数据库中读取数据并进行预处理。预处理过程包括数据包的重组，压缩等，尽量减少数据中的冗余部分。前端程序仅需编写界面的交互逻辑，而不用考虑大流量数据的传输和处理，仅需接收后端程序缓存完成的数据包即可，减轻了浏览器的工作负担，同时简化了程序代码，使其能更快的加载和运行。

4.1.2软件架构和运行逻辑

整个软件系统拥有两个主进程，分别运行于服务器和客户端浏览器，每个主进程中包含若干子线程，共同维护软件的运作。服务器进程需要拥有对接联接管理系统的上行接口，此处通过调用与联接管理系统共享信息的华为云OBS对象储存服务API来实现，这种方案的优势将在下文描述。服务器进程也需要对客户端进程发送的不同HTTP请求进行监听，并返回对应的数据包，所以服务器进程也需要对来自上行接口的数据进行重新整理和压缩，降低数据包大小以节省网络带宽。我们使用json格式的字符串作为服务器进程的通用数据传输格式，即向上行API获取的数据和向客户端发送的数据都以json格式编码。总体上来说，服务器进程可以分为三个子线程，线程1按照一定的周期向上行API调取原始数据，调取到的原始数据以文件形式保存在本地。线程2按照一定的周期对原始数据进行重新整理和压缩，这些处理过后的数据也以文件形式保存在本地，在客户端请求到来时作为HTTP的响应内容发送给客户端。线程3负责监听网络中的HTTP请求，并返回数据包至客户端。三个线程以异步形式运行，线程3开始运行的时间是一个随机值，取决于客户端何时返回请求。线程1和2的工作实际上是一个连续过程，但此处将其分为两个独立线程，原因是当线程1因为数据量过大或网络故障发生阻塞，线程2仍可以保证在一个运行周期过后返回最新数据的处理结果。

客户端进程的执行文件在网页加载初期被发送到用户浏览器上，同时传送的还有网页的静态文件，包括html文档，CSS文件，图片等。之后用户浏览器会运行客户端进程，解析网页静态文件，并根据用户的不同操作向服务器发送不同的HTTP请求。客户端进程中同样拥有异步运行的独立线程，如在网页加载初期，渲染网页静态文档和向服务器请求原始数据由两个分立的线程负责，这种架构保证了网站运行的流畅性。

4.1.3开发平台与框架

随着软件工程技术的发展，越来越多的软件通过不同的开发平台和框架实现，开发平台和框架是对一类形式的软件的抽象，它规定了一种文件结构形式或代码样式，使得在此基础上开发软件更加快速方便，同时拥有更好的性能。同样，物联网系统的客户端应用软件同样基于一些开发平台和框架实现：服务器端与客户端都基于node.js[23]框架开发，node.js是一个开源与跨平台的 JavaScript 运行环境，可以不依赖浏览器运行JavaScript程序，这样一来，服务器端和客户端都可以用JavaScript一种语言来编写，通过标准的node.js接口通信，大大简化了系统的复杂程度。此外，客户端也使用了express[24]和bootstrap[25]框架用于网页交互的设计，实现了高性能的网页渲染和生动的交互效果。此外，客户端的数据可视化图表使用了由百度公司开源的 ECharts[26] 技术，基于此技术，可以以低成本实现惊艳的数据可视化效果。

4.2 应用功能设计

工业设备物联网客户端拥有基本的设备参数监控功能与数据分析结果查看功能。在网页设计中，我们将这两个主要功能分为两个界面，分别是“设备实时参数”和“生产过程记录”。图17为网站的总体页面设计图。

图17 网站页面设计图

其中“主页”页面可用于放置介绍企业基本信息的静态图片或文字（图xx），同时带有通向“设备实时参数”和“生产过程记录”页面的链接。“设备实时参数”页面（图xx）中放置有显示设备参数曲线的图表组件，可以自行编辑和缩放，并带有基本的统计功能。“生产过程记录”页面（图xx）根据相应规则将时序的生产数据分为多个记录集，以列表形式排列在页面上，通过点击子记录按钮，可以进入“记录内容”页面（图18）查看该记录的详细信息，包括相应设备参数曲线，以及系统计算得出的各种数据分析结果。

图18 主页页面

4.2.1设备监控功能

通过“设备实时参数”页面我们可以对联网设备实现实时监控，界面中有多个独立的图表组件，每一个图表都对应一台设备，图表中有多条曲线，可以显示多种设备变量随时间变化的情况。用户可以对图表进行简单操作，如缩放时间轴，隐藏或显示变量曲线，点击数据点查看更多内容等。同时，每个图表组件都可以自由拖拽变换位置，用户可以将两个相同类型设备的监控图排列在同一区域，便于用户对比分析。

图19 设备实时参数页面

4.2.1分析记录查看功能

通过“生产过程记录”页面，我们可以按照日期查找当天的生产记录，通过点击界面上的日历组件，随后系统会自动返回对应日期的生产记录，后台分析程序通过分析当日的设备参数曲线，自动找出不同生产阶段的分界点，再根据这些分界点将一天的生产过程分为多条记录。用户可以点击每条记录后的“查看”按钮进入记录内容界面，该界面会显示更多详细信息供用户查看。

图20 生产记录页面（1）

图21 生产记录页面（2）

4.2 应用实现方案

4.2.1基于node.js/Express技术的网站开发

WEB应用的后端进程运行在网络服务器上，通过监听特定端口传回的客户端HTTP报文，来处理客户端的请求，之后再通过HTTP协议回复客户端。运行在服务器端的后端进程可能需要在同一时间处理多个客户端发来的请求，这就意味着需要在同一时间处理若干个并发连接。此外，后端进程应该在处理请求的空闲时间缓存来自上级数据库的数据，避免在客户端提出请求时再进行数据库读取，从而延长响应时间。以上问题对后端进程的使用性能都有着很大的影响，需要在设计时得到妥善解决。

传统的后端程序代码使用java或python等高级语言编写，开发者需要自行实现用于处理异步并发响应的多线程算法，无论是使用阻塞线程或引入管理线程来处理并发事件，实现起来都较为繁琐，受限于开发者有限的编程水平，这类算法很容易出错且难以达到很好的性能。

为了克服以上难题，我们选择Node.js技术开发后端应用。Node.js是一个开源与跨平台的JavaScript运行环境，拥有独立的Chrome V8引擎，使其可以不依赖浏览器独立运行JavaScript程序[23]。由于JavaScript是一门专门为网络编程创建的编程语言，使用JavaScript开发服务器端程序有着天然的优势，很多专为网络编程设计的语法特性和函数可以直接使用，而无需像其他高级语言那样需要调用特定的插件和库。此外，相比传统开发方式，Node.js拥有更加强大的异步并发处理能力，同时支持各种最新的网络编程技术。Node.js异步运行的特点贯穿于整个编程过程中，如大量的使用回调函数，使得它再执行I/O操作（网络读取、访问数据库或文件系统）时，会在响应返回时恢复操作，而不是阻塞线程使得CPU循环等待。

Express 是一个保持最小规模的灵活的 Node.js Web 应用程序开发框架，为 Web 和移动应用程序提供一组强大的功能[27]。Express框架集成了许多中间件可直接供用户使用，例如解析json文件、解析cookie、解析日志等工作都可以通过调用中间件来完成。使用Express框架来开发网站可以大大减轻开发者的负担，开发者可以把工作重心完全放在实现网站核心功能算法上。

4.2.2程序架构

本客户端软件按照标准的Node.js网页应用架构设计，代码文件包括运行在客户浏览器上的界面脚本，以及运行于服务器端的路由控制脚本，服务器程序由一个独立的启动脚本启动，图22显示了本软件的代码文件结构：

图22 客户端应用代码文件结构示意图

其中字段index代表“主页”，page1A代表“设备实时参数”页面，page2A代表“生产过程记录”页面，page1B代表“记录内容”页面。我们设置了一个public文件夹来存放网站的静态文件，包括字体、图片、CSS文件和前端代码，这些静态文件对网站访问者开放，所以必须与其他代码文件隔离。此外，我们设置了routes文件夹来存放路由控制器代码。Express会根据前端返回的不同URL来调用对应页面的路由控制器来处理响应，网站后端的大部分逻辑实现都在这些路由控制器代码中。views文件夹单独用于放置html文档，每个html文档都描述了一个页面的组件布局和初始化过程。app.js文件中有整个后端程序的启动入口，以及各种初始化代码。关于app.js的详细内容请参考附录C1

4.2.2后端程序工作模式

后端程序运行于远程服务器，主要负责处理前端发回的各种响应请求，不同请求使用不同的URL来标记，后端主程序通过识别不同的请求URL会交给路由处理来转发给相应的路由控制器（图xx棕褐色代码文件）。 在路由控制器中，Express框架使用回调函数来处理页面请求和响应，通过调用request和response对象中的指定函数来获取收到的请求URL或发回响应数据[27]。 客户端软件在一共放置了四个路由控制器程序，分别处理主页、生产记录、实时参数、记录内容页面发回的不同URL，下表列出了网站URL的路由规则。

表7 网站路由表

URL 目标页面请求描述

/index index 返回主页

/page1A page1A 进入设备参数页面

/page2A page2A 进入生产记录页面

/page1A/data -- 请求实时数据json文件

/page2A/event page1B 进入记录内容界面

/page2A/evData -- 请求生产记录列表json文件

/page2A/evAnalysis -- 请求记录分析结果json文件

图23 后端程序工作模式

图23显示了后端程序的完整工作模式，此处实时数据和分析结果数据由联接管理系统上传至华为云OBS（对象储存服务）服务器中，网站后端程序通过双重加密验证访问OBS服务器，下载所需数据。由于实时数据量巨大，在客户请求时再进行远程读取可能会花费较长的时间，所以我们单独设置了一个缓存程序，通过预先设置的定时器控制，在空闲时间下载实时数据至本地

，在前端访问时直接从本地缓存中读取实时数据（参考附录C2）。在一定的更新延迟，但由于物联网数据的更新多以秒或分钟计算，所以对数据完整性的影响并不大，但这种方式能显著加快网站响应速度，对于提升整体使用体验有很大的帮助。除实时数据是通过缓存形式读取外，其他类型的数据都是现场连接远程服务器读取，因为数据量较小，所以延迟基本可以忽略不计。

4.3.1基于Bootstrap/Echarts的前端开发

在编写前端程序时，我们运用了Bootstrap和jQuery框架来实现更高级的前端交互效果。Bootstrap是Twitter推出的一个用于前端开发的开源工具包。它由Twitter的设计师Mark Otto和Jacob Thornton合作开发,是一个CSS/HTML框架[25]。Bootstrap集成了一系列美观且功能丰富的网页布局组件，可以实现更高级的交互效果。Bootstrap在前端交互中的应用实例如图24所示。

图24 基于Bootstrap实现的视觉效果案例图

本应用借助了百度公司开源的Echarts【】技术来实现前端丰富多样的图表效果。Echarts是百度公司面向数据可视化应用开发而开源的Javascript插件[26]，通过Echarts可以很方便的实现带有高级缩放和反馈功能的图表。受篇幅限制本文不对Echarts技术做过多介绍。

4.3.2前端程序工作模式

界面脚本代码文件与每一个界面html文档对应，负责控制界面中各个图表和组件，以及界面效果加载等，这部分代码在网页加载时被发送到客户机上，随后在客户端浏览器运行，这就要求严格控制代码文件大小并简化内部算法，以适应不同性能的浏览器。我们通过将耗时耗力的数据规整和表格渲染任务被安排在服务器端完成，减轻浏览器的工作压力，但这种方式可能会导致页面加载时间延长，因为浏览器必须等待服务器返回预处理好的结果才能进行画面下一步的渲染，为此，我们设计了一个分段数据传输算法来提升页面加载速度。在页面加载的不同阶段，客户端算法与服务器端算法将会进行多次双向通信，按照数据量大小分批次发送数据文件，客户端进程可在数据传输时同时进行界面渲染，这样大大提升了页面的整体加载速度。

前端工作模式图（图25）描述了每一个页面在浏览器打开时的加载步骤，可以看出在page1A页面加载时，设备实时数据是分段传输并渲染的。此外在其他页面的加载过程中，我们充分应用了node.js框架的异步处理特性，尽量将多个加载和渲染任务并行完成以提高页面的整体加载速度。

图25 前端程序工作模式图

4.4 代码维护与功能拓展

考虑到后期代码的维护与功能拓展问题，在设计网页客户端时，我们充分遵循了MVC设计模式[28]，这种设计模式提倡将用户界面与后端逻辑代码完全分开，并在中间编写一个专用的控制器对象来进行前后端参数的调度，以实现前端页面代码的良好封装，但同时可以通过修改控制器对象来继续利用原有的逻辑函数。

指　　标
疑似剽窃文字表述

1. 它还提供了基于HTTP/HTTPS协议的Web服务接口，用户可以随时随地连接到Internet的电脑上，通过OBS管理控制台或各种OBS工具访问和管理存储在OBS中的数据。
2. Bootstrap和jQuery框架来实现更高级的前端交互效果。Bootstrap是Twitter推出的一个用于前端开发的开源工具包。

4. 工业物联网数据管理信息系统与终端硬件设计_第4部分

总字数：15869

相似文献列表

去除本人文献复制比：15.3%(2435)　　文字复制比：15.3%(2435)　　疑似剽窃观点：(0)

1	源代码(/* USER CODE BEGIN Header */ /** ********** ...) - 《源代码库（https://raw.githubus）》- 2020	11.8%（1874） 是否引证：否
2	2015040206023-辛政方-A Study on Wireless Temperature Sensor Networks for Wildfire Alarm 辛政方 - 《大学生论文联合比对库》- 2019-05-07	11.5%（1829） 是否引证：否
3	源代码(/** ********** ...) - 《源代码库（https://raw.githubus）》- 2020	11.2%（1775） 是否引证：否
4	源代码(/* USER CODE BEGIN Header */ /** ********** ...) - 《源代码库（https://raw.githubus）》- 2020	11.2%（1774） 是否引证：否
5	源代码(/* USER CODE BEGIN Header */ /** ********** ...) - 《源代码库（https://raw.githubus）》- 2020	11.0%（1752） 是否引证：否
6	源代码(/* Copyright (c) 2011 Arduino. All right res...) - 《源代码库（https://raw.githubus）》- 2020	8.5%（1355） 是否引证：否
7	源代码(/* * File : board.c * T) - 《源代码库（https://gitee.com/rt）》- 2018	7.4%（1182） 是否引证：否
8	linux进程监控与自动重启实现 - yangyinbo的专栏 - CSDN博客 - 《网络（http://blog.csdn.net）》- 2017	1.7%（264） 是否引证：否
9	源代码(/** **********) - 《源代码库（https://gitee.com/ma）》- 2018	1.5%（242） 是否引证：否

10	源代码(/** **...)	1.5%（242）
	- 《源代码库（https://raw.githubus）》- 2020	是否引证：否
11	源代码(/**)	1.5%（241）
	- 《源代码库（https://github.com/f）》- 2018	是否引证：否
12	源代码(/** **********************)	1.5%（241）
	- 《源代码库（https://github.com/f）》- 2018	是否引证：否
13	源代码(/** **********************)	1.5%（241）
	- 《源代码库（https://gitee.com/rt）》- 2018	是否引证：否
14	源代码(stm32f4xx_hal_uart.c)	1.5%（241）
	- 《源代码库（https://raw.githubus）》- 2016	是否引证：否
15	源代码(/** **...)	1.5%（241）
	- 《源代码库（https://raw.githubus）》- 2020	是否引证：否
16	源代码(/** **...)	1.5%（241）
	- 《源代码库（https://raw.githubus）》- 2020	是否引证：否
17	源代码(/** **...)	1.5%（241）
	- 《源代码库（https://raw.githubus）》- 2020	是否引证：否
18	源代码(/** **...)	1.5%（241）
	- 《源代码库（https://raw.githubus）》- 2020	是否引证：否
19	源代码(/** **...)	1.5%（241）
	- 《源代码库（https://raw.githubus）》- 2020	是否引证：否
20	源代码(/** **...)	1.5%（241）
	- 《源代码库（https://raw.githubus）》- 2020	是否引证：否
21	源代码(/** **...)	1.5%（241）
	- 《源代码库（https://raw.githubus）》- 2020	是否引证：否

原文内容

的在软件设计过程中，依据这种设计模式，数据远程读取和各种预处理算法的逻辑代码都后端实现，每一个前端页面的控制代码与其他页面几乎没有任何逻辑关联，实现了良好的封装性。当后期开发中需要继续添加新的功能或增加新的前端页面时，由于算法逻辑和各种参数的传递都在后端完成，很容易实现跨页面的分享数据，同时也可以使用原先的功能函数。

参考文献

[1] 美的集团. M.IoT已成为国内首家自主兼备的工业互联网提供商[J].智能城市,2019,5(03):70.

[2] 中国工业物联网云平台产业演进[J].中国工业和信息化,2018(07):58-66.

[3] 谢海琴. 共建共享、共创共赢——卡奥斯COSMOPlat助力企业数字化转型[J].张江科技评论,2021(01):36-38.

[4] 思科推出物联网新品,推动工业解决方案应用[J].智能制造,2016(07):12.

[5] 文晓. 研华：打造物联网整体解决方案[J].自动化博览,2019(11):26-27.

[6] 邹玉龙,丁晓进,王全全. NB-IoT关键技术及应用前景[J].中兴通讯技术,2017,23(01):43-46.

[7] 彭雄根,李新,陈旭奇. NB-IoT技术的发展及网络部署策略研究[J].邮电设计技术,2017(03):58-61.

[8] 孙书鹰,陈志佳,寇超.新一代嵌入式微处理器STM32F103开发与应用[J].微计算机应用,2010,31(12):59-63.

[9] 杨晓艳,陈亮.基于STM32CubeMX的单片机最小系统设计[J].数字技术与应用,2018,36(06):149-150.

[10] Michael M S. Universal asynchronous receiver/transmitter: U.S. Patent 5,140,679[P]. 1992-8-18.

[11] 潘苏皖,杨凯. 基于STM32CubeMX的串口控制研究[J].电子制作,2021(02):15-16+43.

[12] P Corcoran. Two Wires and 30 Years : A Tribute and Introductory Tutorial to the I2C Two-Wire Bus[J]. IEEE Consumer Electronics Magazine, 2013(07): 30-36

[13] 易志明,林凌,郝丽宏,李树靖. SPI串行总线接口及其实现[J].自动化与仪器仪表,2002(06):47-50.

[14] 卢斌. NB-IoT物联网覆盖增强技术探讨[J].移动通信,2016,40(19):55-59.

[15] 李娟,胡晓玲,李自刚. 窄带物联网NB-IOT能耗测试浅析[J].电信网技术,2016(08):65-67

[16] 徐冬冬. LoRa与NB-IoT技术开启物联网新格局[J].科学技术创新,2017(24):116-117.

[17] 朱小襄. ModBus通信协议及编程[J].电子工程师,2005(07):42-44+55.

[18] 周园春,李淼,张建,李晓欧,张飞. 中间件技术综述[J].计算机工程与应用,2002(15):80-82.

[19] 华为技术有限公司. IoTDA使用流程简介[ER/OL] .(2020/12/16).[2021/5/1]. https://support.huaweicloud.com/usermanual-iothub/iot_01_0015.html

[20] 华为技术有限公司. DIS使用流程简介[ER/OL]. (2021/01/30).[2021/5/1]. https://support.huaweicloud.com/usermanual-dis/dis_01_0009.html

[21] 华为技术有限公司. 对象储存服务 OBS功能介绍[ER/OL]. (2021/03).[2021/5/1]. https://www.huaweicloud.com/product/obs/features.html

[22] 王剑秋,赵一. 物联网传输协议MQTT与CoAP比较与应用[J].计算机时代,2017(10):25-28+31.

[23] 王金龙,宋斌,丁锐. Node.js:一种新的Web应用构建技术[J].现代电子技术,2015,38(06):70-73.

[24] 王伶俐,张传国. 基于NodeJS+Express框架的轻应用定制平台的设计与实现[J].计算机科学,2017,44(S2):596-599.
[25] 傅翠玉,王少茹,洪秀金. Bootstrap框架在响应式WEB开发中的应用[J].电脑知识与技术,2018,14(21):85-86.
[26] 冀潇,李杨. 采用ECharts可视化技术实现的数据体系监控系统[J].计算机系统应用,2017,26(06):72-76.
[27] 程桂花,沈炜,何松林,张珂杰. Node.js中Express框架路由机制的研究[J].工业控制计算机,2016,29(08):101-102.
[28] 毕建信. 基于MVC设计模式的Web应用研究与实现[D].武汉理工大学,2006.

附录

附录A

数据采集终端MCU源代码

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "dma.h"
#include "usart.h"
#include "spi.h"
#include "tim.h"
#include "gpio.h"
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "stdio.h"
#include "string.h"
#include "oled.h"
/* USER CODE END Includes */
void SystemClock_Config(void);
uint16_t Modbus_Slave_ID = 0x01; //定义Modbus Slave ID 默认0x01
uint16_t Modbus_Hold_Reg[16]; //Modbus Hold Register
#define UART1_RX_LEN 1024 //UART1接收缓存
uint8_t UART1_RX_BUF[UART1_RX_LEN];
__IO uint16_t UART1_RX_STA = 0;
#define UART3_RX_LEN 1024 //UART2接收缓存
uint8_t UART3_RX_BUF[UART3_RX_LEN];
__IO uint16_t UART3_RX_STA = 0;
uint16_t AT_command = 0;
uint16_t text8y = 0;
void OLED_CLR(){ //OLED清屏
OLED_Clear();
text8y = 0;
OLED_Refresh();
}
void TrimSpace(char* str) //OLED字符去除空格和换行符
{
char *start = str - 1;
char *end = str;
char *p = str;
while(*p)
{
switch(*p)
{
case ' ':
{
if(start + 1==p)
start = p;
}
break;
case '\r':
{
if(start + 1==p)
start = p;
}
break;
case '\n':
{
if(start + 1==p)
```

```
start = p;
}
break;
default:
break;
}
++p;
}
--p;
++start;
if(*start == 0)
{
*str = 0 ;
return;
}
end = p + 1;
while(p > start)
{
switch(*p)
{
case ' ':
{
if(end - 1 == p)
end = p;
}break;
case '\r':
{
if(end - 1 == p)
end = p;
}break;
case '\n':
{
if(end - 1 == p)
end = p;
}
break;
default:
break;
}
--p;
}
memmove(str,start,end-start);
*(str + (int)end - (int)start) = 0;
}
void LED_Bleep(int ms, int times)
{
for(int i = 0; i<times*2; i++)
{
HAL_GPIO_TogglePin(LED_GPIO_Port,LED_Pin);//LED亮灭状濁翻轿
HAL_Delay(ms);//延时1000毫秒=1秒钟
}
HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
}
void USART1_IRQHandler(void)
{
//检查各种异常中断行为
if(__HAL_UART_GET_FLAG(&huart1, UART_FLAG_ORE )!= RESET)
{
__HAL_UART_CLEAR_OREFLAG(&huart1); //去除ORE中断位
}else if(__HAL_UART_GET_FLAG(&huart1, UART_FLAG_NE )!= RESET){
__HAL_UART_CLEAR_NEFLAG(&huart1); //去除NE中断位
```

```
}else if(__HAL_UART_GET_FLAG(&huart1, UART_FLAG_FE )!= RESET){
__HAL_UART_CLEAR_FEFLAG(&huart1); //去除FE中断位
}else if(__HAL_UART_GET_FLAG(&huart1, UART_FLAG_PE )!= RESET){
__HAL_UART_CLEAR_PEFLAG(&huart1); //去除PE中断位
}else{
}
if(__HAL_UART_GET_FLAG(&huart1, UART_FLAG_IDLE) != RESET) //若空闲中断标记被置位
{
__HAL_UART_CLEAR_IDLEFLAG(&huart1); // 清楚中断位
HAL_UART_DMAStop(&huart1);
UART1_RX_STA = UART1_RX_LEN - __HAL_DMA_GET_COUNTER(huart1.hdmarx); // 总数据量减去未接收到的数据量为已经接收到的数据量
UART1_RX_BUF[UART1_RX_STA] = 0; // 添加结束答
UART1_RX_STA |= 0X8000; // 标记接收结束
HAL_UART_Receive_DMA(&huart1, UART1_RX_BUF, UART1_RX_LEN); // 重启DMA接收
}
}
void USART3_IRQHandler(void)
{
if(__HAL_UART_GET_FLAG(&huart3, UART_FLAG_ORE )!= RESET)
{
//printf("ORE\r\n");
__HAL_UART_CLEAR_OREFLAG(&huart3);
}else if(__HAL_UART_GET_FLAG(&huart3, UART_FLAG_NE )!= RESET){
//printf("NE \r\n");
__HAL_UART_CLEAR_NEFLAG(&huart3);
}else if(__HAL_UART_GET_FLAG(&huart3, UART_FLAG_FE )!= RESET){
//printf("FE \r\n");
__HAL_UART_CLEAR_FEFLAG(&huart3);
}else if(__HAL_UART_GET_FLAG(&huart3, UART_FLAG_PE )!= RESET){
//printf("PE \r\n");
__HAL_UART_CLEAR_PEFLAG(&huart3);
}else{
//printf("\r\n Normal, ");
}
if(__HAL_UART_GET_FLAG(&huart3, UART_FLAG_IDLE) != RESET) //若空闲中断标记被置位
{
//printf("Interrupt reset\r\n");
__HAL_UART_CLEAR_IDLEFLAG(&huart3); // 清楚中断标记
HAL_UART_DMAStop(&huart3); // 停止DMA接收
UART3_RX_STA = UART3_RX_LEN - __HAL_DMA_GET_COUNTER(huart3.hdmarx); // 总数据量减去未接收到的数据量为已经接收到的数据量
//printf("PC: UART3_RX_STA: %u\r\n", UART3_RX_STA);
UART3_RX_BUF[UART3_RX_STA] = 0; // 添加结束答
UART3_RX_STA |= 0X8000; // 标记接收结束
HAL_UART_Receive_DMA(&huart3, UART3_RX_BUF, UART3_RX_LEN); // 重启DMA接收
}
}
//Modbus 03 读取保持寄存嚣16使
void Read_Reg_Hold(void)
{
//printf("reg\r\n");
if((UART1_RX_STA & 0X7FFF) > 0x06)
{
if(UART1_RX_BUF[2]==0x00 && UART1_RX_BUF[4]==0x00)
{
uint16_t start_reg = UART1_RX_BUF[3]; //起始寄存器地坍
uint16_t bit_num = UART1_RX_BUF[5]; //读取寄存器数釿
printf("PC: START_REG_NUM: %u, BIT_NUM %u \r\n",start_reg,bit_num);
}
}
```

```
}
void Modbus_Handle(void)
{
//HAL_UART_Transmit(&huart1, UART1_RX_BUF, UART1_RX_STA & 0X7FFF, 100); // ???????????
printf("PC: MODBUS HEX, LENGTH: %u \r\n", UART1_RX_STA & 0X7FFF);
if(UART1_RX_BUF[0]!=Modbus_Slave_ID)
{
printf("PC: ID is not correct.\r\n");
}
else{
printf("PC: ID is correct.\r\n");
switch(UART1_RX_BUF[1])
{
case 0x01:{}break; //读取输出线圈
case 0x02:{}break; //读取输入线圈
case 0x03: //读取保持寄存器
{
Read_Reg_Hold();
}break;
case 0x04:{}break; //读取输入寄存器
case 0x05:{}break; //设置单个线圈
case 0x06:{}break; //设置单个寄存器
case 0x0f:{}break; //设置多个线圈
case 0x10:{}break; //设置多个寄存器
default:{}break; //错误
}
}
}
void AT_Command_Set(int step)
{
switch(step)
{
case 1:{
OLED_CLR();
printf("\r\nPC: AT+CGSN=1------\r\n");
AT_command = 1;
uint8_t * net_test = (uint8_t*)"AT+CGSN=1\r\n";
HAL_UART_Transmit(&huart3, net_test, 16, 100); // ???????????
}break;
case 2:{
printf("\r\nPC: AT+CGATT?---------\r\n");
AT_command = 2;
uint8_t * net_test = (uint8_t*)"AT+CGATT?\r\n";
HAL_UART_Transmit(&huart3, net_test, 16, 100); // ???????????
}break;
case 3:{
printf("\r\nPC: AT+CSQ---------\r\n");
AT_command = 3;
uint8_t * net_test = (uint8_t*)"AT+CSQ\r\n";
HAL_UART_Transmit(&huart3, net_test, 16, 100); // ???????????
}break;
case 4:{
printf("\r\nPC: AT+NMGS---------\r\n");
AT_command = 4;
uint8_t * net_test = (uint8_t*)"AT+NMGS=17,0019992f4b4ccc66667fff7289b332ffff\r\n";
HAL_UART_Transmit(&huart3, net_test, 64, 100); // ???????????
}
default:{}break;
}
}
//AT指令初始化
```

```
void AT_RX_Handle(void)
{
TrimSpace((char*)UART3_RX_BUF);
HAL_UART_Transmit(&huart1, UART3_RX_BUF, UART3_RX_STA & 0X7FFF, 100); // ???????????
UART3_RX_STA = 0;
OLED_ShowString(0,text8y,UART3_RX_BUF,8,1);
text8y+=8;
OLED_Refresh();
switch(AT_command)
{
case 1:{
AT_Command_Set(2);
}break;
case 2:{
AT_Command_Set(3);
}break;
case 3:{
AT_Command_Set(4);
}break;
default:{}break;
}
OLED_Refresh();
}
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
OLED_CLR();
AT_Command_Set(1);
}
/* USER CODE END 0 */
/**
* @brief The application entry point.
* @retval int
*/
int main(void)
{
HAL_Init();
SystemClock_Config();
MX_GPIO_Init();
MX_DMA_Init();
MX_USART1_UART_Init();
MX_USART3_UART_Init();
MX_LPUART1_UART_Init();
MX_SPI1_Init();
MX_TIM2_Init();
OLED_Init();
LED_ON;
OLED_Refresh();
HAL_TIM_Base_Start_IT(&htim2);
//USART1 Modbus串口初始化
HAL_UART_Receive_DMA(&huart1, UART1_RX_BUF, UART1_RX_LEN);
__HAL_UART_ENABLE_IT(&huart1, UART_IT_IDLE);
__HAL_UART_ENABLE_IT(&huart3, UART_IT_IDLE); //使能空闲中断
/* USER CODE BEGIN WHILE */
while (1)
{
//若USART1 Modbus串口接收到数据
if(UART1_RX_STA & 0X8000)
{
printf("PC: Modbus handler \r\n");
Modbus_Handle();
UART1_RX_STA = 0;
```

```
}
//若UART3 Modbus串口接收到数据
if(UART3_RX_STA & 0X8000)
{
//printf("PC: AT handler \r\n");
AT_RX_Handle();
}
}
/* USER CODE END 3 */
}
void SystemClock_Config(void)
{
RCC_OscInitTypeDef RCC_OscInitStruct = {0};
RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};
/** Initializes the CPU, AHB and APB busses clocks
*/
RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
RCC_OscInitStruct.HSEState = RCC_HSE_ON;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
{
Error_Handler();
}
/** Initializes the CPU, AHB and APB busses clocks
*/
RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSE;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
{
Error_Handler();
}
PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_USART1|RCC_PERIPHCLK_USART3
|RCC_PERIPHCLK_LPUART1;
PeriphClkInit.Usart1ClockSelection = RCC_USART1CLKSOURCE_PCLK2;
PeriphClkInit.Usart3ClockSelection = RCC_USART3CLKSOURCE_PCLK1;
PeriphClkInit.Lpuart1ClockSelection = RCC_LPUART1CLKSOURCE_PCLK1;
if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
{
Error_Handler();
}
/** Configure the main internal regulator output voltage
*/
if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
{
Error_Handler();
}
}
/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
if(GPIO_Pin==KEY1_Pin)
{
OLED_CLR();
AT_Command_Set(1);
}
if(GPIO_Pin==KEY2_Pin)
```

```
{
OLED_CLR();
AT_Command_Set(3);
}
}
void Error_Handler(void)
{
//异常处理空置
}
```

附录B

联接管理系统主要源代码

附B1

主程序源代码(Service.py)

```
# !/usr/bin/python
# -*- coding:utf-8 -*-
import DataFromDIS
from RealTimeDataPool import RealTimeDataPool
from AutoClaveBreakStorage import AutoClaveBreakStorage
from AutoClaveDataAnalysis import AutoClaveDataAnalysis
import time
import json
import schedule
from obs import ObsClient
from configparser import ConfigParser
#version 10/31
para = 1
period = 5
conf_path = 'conf.ini'
def new_obs_client():
conf = ConfigParser()
conf.read(conf_path)
# Use configuration file
try:
ak = conf.get('OBSconfig', 'ak')
sk = conf.get('OBSconfig', 'sk')
server = conf.get('OBSconfig', 'server')
obs_client = ObsClient(access_key_id=ak, secret_access_key=sk, server=server)
return obs_client
except Exception as ex:
print('New OBS client ' + str(ex))
pass
def job():
modbus_record = DataFromDIS.get_records()
obs_client = new_obs_client()
real_time_data_pool = RealTimeDataPool(obs_client)
np_data_list = real_time_data_pool.auto_clave_process(modbus_record)
break_storage = AutoClaveBreakStorage(obs_client, np_data_list)
resp, now = break_storage.break_storage_process()
analysis_obj = AutoClaveDataAnalysis(obs_client, now)
resp = analysis_obj.data_analysis_process()
if para == 1:
job()
elif para == 2:
print('nojob')
obs_client = new_obs_client()
bucket_name = 'obs-ydyl'
prefix = 'Service/ZyRecord/2020-10-17/4XFIN1602902895Y1602943935'
resp = obs_client.getObject(bucket_name, prefix, downloadPath=None)
print(resp)
records = ''
if resp.status < 300:
```

```
response = resp.body.response
chunk_size = 65536
if response is not None:
while True:
chunk = response.read(chunk_size)
if not chunk:
break
records = records+bytes.decode(chunk)
response.close()
print(json.loads(records))
else:
schedule.every(period).minutes.do(job)
while True:
schedule.run_pending()
time.sleep(1)
```

附B2

实时数据处理程序源代码(RealTimeDataPool.py)

```
import json
from configparser import ConfigParser
from datetime import datetime
import numpy as np
confPath = 'conf.ini'
bucket_name = 'obs-ydy1'
def auto_clave_data_convert(data_json, obs_client):
conf = ConfigParser()
conf.read(confPath)
clave_num = 7
np_array = []
for clave_id in range(1, clave_num + 1):
dev_id = conf.get('AutoClave' + str(clave_id), 'devId')
in_temp_channel = conf.get('AutoClave' + str(clave_id), 'inTempChannel')
in_temp_slope = conf.get('AutoClave' + str(clave_id), 'inTempSlope')
in_temp_shift = conf.get('AutoClave' + str(clave_id),'inTempShift')
out_temp_channel = conf.get('AutoClave' + str(clave_id), 'outTempChannel')
out_temp_slope = conf.get('AutoClave' + str(clave_id), 'outTempSlope')
out_temp_shift = conf.get('AutoClave' + str(clave_id), 'outTempShift')
in_press_channel = conf.get('AutoClave' + str(clave_id), 'inPressChannel')
in_press_slope = conf.get('AutoClave' + str(clave_id), 'inPressSlope')
in_press_shift = conf.get('AutoClave' + str(clave_id), 'inPressShift')
state_channel = conf.get('AutoClave' + str(clave_id), 'stateChannel')
time_list = []
in_temp_list = []
out_temp_list = []
in_press_list = []
state_list = []
record_dict = []
for devRec in data_json:
data = json.loads(devRec['data'])
try:
device_id = data['notify_data']['header']['device_id']
if device_id == dev_id:
services = data['notify_data']['body']['services'][0]
properties = services['properties']
try:
time = datetime.strptime(services['event_time'], '%Y%m%dT%H%M%SZ').timestamp() + 3600 * 8
in_temp = float(properties[in_temp_channel]) * float(in_temp_slope) + float(in_temp_shift)
out_temp = float(properties[out_temp_channel]) * float(out_temp_slope) + float(out_temp_shift)
in_press = float(properties[in_press_channel]) * float(in_press_slope) + float(in_press_shift)
state = properties[state_channel]
time_list.append(time)
in_temp_list.append(in_temp)
```

```
    out_temp_list.append(out_temp)
    in_press_list.append(in_press)
    state_list.append(state)
    rec_dict = {'time': time, 'inTemp': in_temp,
    'outTemp': out_temp, 'inPress': in_press,
    'state': state}
    record_dict.append(rec_dict)
    except:
    pass
    except:
    pass
    data_set_np = np.array([time_list, in_temp_list, out_temp_list, in_press_list, state_list])
    np_array.append(data_set_np)
    obs_rec_dict = json.dumps({'claveId': clave_id, 'records': record_dict})
    obs_rec_prefix = 'Service/ZyRealTime/clave' + str(clave_id)
    resp = obs_client.deleteObject(bucket_name, obs_rec_prefix)
    if resp.status >= 300:
    print('[RealTimeDataPool] OBS DELETE obj: common msg:status:', resp.status, 'prefix ', obs_rec_prefix,
',errorCode:',
    resp.errorCode, ',errorMessage:', resp.errorMessage)
    resp = obs_client.putContent(bucket_name, obs_rec_prefix, str(obs_rec_dict))
    if resp.status >= 300:
    print('[RealTimeDataPool] OBS PUT content: common msg:status:', resp.status, 'prefix ', obs_rec_prefix,
',errorCode:',
    resp.errorCode, ',errorMessage:', resp.errorMessage)
    return np_array
    class RealTimeDataPool:
    def __init__(self, obs_client):
    self.obs_client = obs_client
    pass
    def auto_clave_process(self, data_json):
    return auto_clave_data_convert(data_json, self.obs_client)
    附B3
    DIS连接程序源代码(DataFromDIS.py)
    from src.com.dis.client import disclient
    from configparser import ConfigParser
    confPath = 'conf.ini'
    partition_id = 'shardId-0000000000'
    startSeq = 0
    streamName = 'dis-YDY1'
    def get_cursor_test(cli):
    try:
    r = cli.getCursor(streamName=streamName, partitionId=partition_id, cursorType='TRIM_HORIZON',
startSeq=startSeq)
    return r.cursor
    except Exception as ex:
    print('DISDataMan __getCursor_test ' + str(ex))
    def new_object():
    conf = ConfigParser()
    conf.read(confPath)
    # Use configuration file
    try:
    project_id = conf.get('DISconfig', 'projectid')
    ak = conf.get('DISconfig', 'ak')
    sk = conf.get('DISconfig', 'sk')
    region = conf.get('DISconfig', 'region')
    endpoint = conf.get('DISconfig', 'endpoint')
    try:
    dis = disclient.disclient(endpoint=endpoint, ak=ak, sk=sk, projectid=project_id, region=region)
    return dis
    except Exception as ex:
```

```
print('[ZYFDataFromDIS] (new_object) dislink' + str(ex))
except Exception as ex:
print('[ZYFDataFromDIS] (new_object) conf load ' + str(ex))
pass
def get_records():
cli = new_object()
cursor = get_cursor_test(cli)
records = []
try:
while cursor:
r = cli.getRecords(partitioncursor=cursor)
cursor = r.nextPartitionCursor
if not r.recordResult:
break
records.extend(r.body["records"])
return records
except Exception as ex:
print('[ZYFDataFromDIS] (get_records_test)' + str(ex))
```

附B4

联接管理系统启动脚本 (autostart.shell)

```
#!/bin/sh
host_dir=`echo ~~ # 当前用户根目录
proc_name="/home/Iot-CMP/service.py" # 进程名
file_name="/home/Iot-CMP/log/autoRestart.log" # 日志文件
pid=0
proc_num() # 计算进程数
{
num=`ps -ef | grep $proc_name | grep -v grep | wc -l`
return $num
}
proc_id() # 进程号
{
pid=`ps -ef | grep $proc_name | grep -v grep | awk '{print $2}'`
}
proc_num
number=$?
if [ $number -eq 0 ] # 判断进程是否存在
then
nohup python3 /home/Iot-CMP/service.py & # 重启进程
proc_id # 获取新进程号
echo ${pid},
```



```
`date` >> $file_name # 将新进程号和重启时间记录
fi
```

附B5

联接管理系统设备配置文件(configuration.conf)

```
[DISconfig]
projectid=092b1a7e3380f2172f28c01d4f6a3fcf
ak=C6YFHVZERH6SNUS3T8RX
sk=g6yPZO5O4EsfgJFbYNwxp3DZ7wmGg5pSSIYtLSLe
region=cn-north-4
endpoint=dis.cn-north-4.myhuaweicloud.com
```

```
[OBSconfig]
ak=C6YFHVZERH6SNUS3T8RX
sk=g6yPZO5O4EsfgJFbYNwxp3DZ7wmGg5pSSIYtLSLe
region=cn-north-4
server=https://obs.cn-north-4.myhuaweicloud.com
[AutoClave1]
devId=5f9d05fcadf72402d0d9ecb6_01
inTempChannel=ch5
inTempSlope=2.5044
inTempShift=-260.0353
outTempChannel=ch4
outTempSlope=0.1
outTempShift=0
inPressChannel=ch6
inPressSlope=0.0006
inPressShift=-0.1026
stateChannel=ch0
[AutoClave2]
devId=5f9d05fcadf72402d0d9ecb6_02
inTempChannel=ch5
inTempSlope=2.5044
inTempShift=-260.0353
outTempChannel=ch4
outTempSlope=0.1
outTempShift=0
inPressChannel=ch6
inPressSlope=0.0006
inPressShift=-0.1026
stateChannel=ch0
[AutoClave3]
devId=5f9d05fcadf72402d0d9ecb6_03
inTempChannel=ch5
inTempSlope=2.5044
inTempShift=-260.0353
outTempChannel=ch4
outTempSlope=0.1
outTempShift=0
inPressChannel=ch6
inPressSlope=0.0006
inPressShift=-0.1026
stateChannel=ch0
[AutoClave4]
devId=5f9d05fcadf72402d0d9ecb6_04
inTempChannel=ch5
inTempSlope=2.5044
inTempShift=-260.0353
outTempChannel=ch4
outTempSlope=0.1
outTempShift=0
inPressChannel=ch6
inPressSlope=0.0006
inPressShift=-0.1026
stateChannel=ch0
[AutoClave5]
devId=5f9d05fcadf72402d0d9ecb6_05
inTempChannel=ch5
inTempSlope=2.5044
inTempShift=-260.0353
outTempChannel=ch4
outTempSlope=0.1
outTempShift=0
```

```
inPressChannel=ch6
inPressSlope=0.0006
inPressShift=-0.1026
stateChannel=ch0
[AutoClave6]
devId=5f9d05fcadf72402d0d9ecb6_06
inTempChannel=ch5
inTempSlope=2.5044
inTempShift=-260.0353
outTempChannel=ch4
outTempSlope=0.1
outTempShift=0
inPressChannel=ch6
inPressSlope=0.0006
inPressShift=-0.1026
stateChannel=ch0
[AutoClave7]
devId=5f9d05fcadf72402d0d9ecb6_07
inTempChannel=ch5
inTempSlope=1.2686
inTempShift=-249.74
outTempChannel=ch4
outTempSlope=0.1
outTempShift=0
inPressChannel=ch6
inPressSlope=0.0006
inPressShift=-0.1026
stateChannel=ch0
```

附录C

客户端软件源代码

附C1

后端启动程序源代码（app.js）

```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');
var ejs = require('ejs');
var compression = require('compression');
var app = express();
app.use(compression());
app.set('views', path.join(__dirname, 'views'));
app.engine('html', ejs.renderFile);
app.set("view engine", "html");
// 加载日志中间件
app.use(logger('dev'));
// 加载解析json的中间件
app.use(express.json());
// 加载解析urlencoded请求体的中间件
app.use(express.urlencoded({ extended: false }));
// 加载解析cookie的中间件
app.use(cookieParser());
// 设置public文件夹为存放静态文件的目录
app.use(express.static(path.join(__dirname, 'public')));
//参数‘/’可当作设置url的根显示页面，这里即”http://localhost:3000/“访问的页面设置为index.html
app.get('/',(req,res)=>{
res.sendFile(__dirname+"/views/"+"zyrc.html") //设置/ 下访问文件位置
});
// 路由控制器
var indexRouter = require('./routes/index');
var zyfRouter = require('./routes/zyf');
```

```
var zyrcRouter = require('./routes/zyrc');
app.use('/', indexRouter);
app.use('/zyf', zyfRouter);
app.use('/zyrc', zyrcRouter);
app.use('/zyrcEv', zyrcRouter);
// 捕获404错误，并转发到错误处理器
app.use(function(req, res, next) {
next(createError(404));
});
// error handler
app.use(function(err, req, res, next) {
// set locals, only providing error in development
res.locals.message = err.message;
res.locals.error = req.app.get('env') === 'development' ? err : {};
// render the error page
res.status(err.status || 500);
res.render('error');
});
// 导出app实例，供其他模块调用
module.exports = app;
var server =app.listen(3000,()=>{
var port =server.address().port
console.log("【】访问地址http://localhost:%s",port)
});
//ScheduleProc.startScheduleCycle();
```

附C2

后端“实时数据”界面路由源代码（page1A.js）

```
var express = require('express');
var path = require('path');
const {resolve} = require('path');
var ejs = require('ejs');
const fs = require('fs');
var OBSkit = require(resolve(__dirname,'../service/OBSkit'));
var gb = require('json-groupby');
var app = express();
app.set('views', path.join(__dirname, 'views'));
app.engine('html', ejs.renderFile);
app.set("view engine", "html");
/* GET users listing. */
app.get('/', function(req, res, next) {
console.log('sending file');
res.sendFile(resolve(__dirname, '../views/zyf.html'));
//res.json(jsonToClient);
});
app.get('/datafu', function (req, res, next) {
var FuId = req.query.FuId;
var prefix = "Service/ZyRealTime/clave"+FuId;
OBSkit.getStr(prefix,0,function(contents,prefix,para){
var jsdata = JSON.parse(contents);
res.json(jsdata);
});
});
module.exports = app;
```

附C3

后端“生产记录”界面路由源代码（page2A.js）

```
var express = require('express');
var path = require('path');
const {resolve} = require('path');
var ejs = require('ejs');
var OBSkit = require(resolve(__dirname,'../service/OBSkit'));
var app = express();
```

```
var informJson = {
FuId:0,
startTime:0,
endTime:0,
prefix:'',
data:''
}
app.set('views', path.join(__dirname, 'views'));
app.engine('html', ejs.renderFile);
app.set("view engine", "html");
/* GET users listing. */
app.get('/', function(req, res, next) {
res.sendFile(resolve(__dirname, '../views/zyrc.html'));
});
app.get('/zyev', function(req, res, next) {
informJson.FuId = req.query.id;
informJson.startTime = req.query.startTime;
informJson.endTime = req.query.endTime;
informJson.prefix = req.query.prefix;
console.log("Prefix="+req.query.prefix);
res.sendFile(resolve(__dirname, '../views/zyrcEv.html'));
});
app.get('/zyevId', function(req, res, next) {
var para_A;
//console.log('zyevID Prefix:'+informJson.prefix);
OBSkit.getStr(informJson.prefix,para_A,function(contents,prefix,para_B){
var recordJson = JSON.parse(contents);
informJson.data = recordJson.data;
res.json(informJson);
});
});
app.get('/list', function(req, res, next) {
var searchDate = req.query.date;
var prefix = 'Service/ZyRecord/'+searchDate+"/";
console.log("SearchDate Prefix: "+prefix);
OBSkit.OBSFolderObj(prefix,app,function(contents,prefix,router){
var eventListJson = [];
var smallTime = 0;
var index = 1;
var maxInt = 1000;
var inter = 1;
if(contents.length>1){
while(eventListJson.length<contents.length && inter<maxInt){
inter = inter+1;
var bigTime = 1000000000000000;
var evPrefix = '';
console.log('for')
for(let j=0;j<contents.length;j++){
var eventPrefix = contents[j]['Key'];
console.log(eventPrefix)
var Xindex = eventPrefix.indexOf('X');
if(Xindex>0){
var Yindex = eventPrefix.indexOf('Y');
var startTimeTemp = parseInt(eventPrefix.substring(Xindex+4,Yindex))*1000;
if(startTimeTemp<bigTime && startTimeTemp>smallTime){
bigTime = startTimeTemp;
evPrefix = eventPrefix;
}
}
}
smallTime = bigTime;
```

```
var Xindex_A = evPrefix.indexOf('X');
if(Xindex_A>0){
var Yindex_A = evPrefix.indexOf('Y');
var startTime = parseInt(evPrefix.substring(Xindex_A+4,Yindex_A))*1000;
var endTime = parseInt(evPrefix.substring(Yindex_A+1))*1000;
var fuId = evPrefix.substr(Xindex_A-1,1);
var recordJson = {
index : index,
fuId : fuId,
startTime : startTime,
endTime : endTime,
prefix:evPrefix
}
eventListJson.push(recordJson);
index = index+1;
}
}
}
res.json(eventListJson);
console.log(eventListJson)
console.log('sended');
});
});
module.exports = app;
```

附C4

后端“主页”页面路由代码（index.js）

```
var express = require('express');
var path = require('path');
const {resolve} = require('path');
var ejs = require('ejs');
var app = express();
app.set('views', path.join(__dirname, 'views'));
app.engine('html', ejs.renderFile);
app.set("view engine", "html");
/* GET users listing. */
app.get('/', function(req, res, next) {
res.sendFile(resolve(__dirname, '../views/index.html'));
});
module.exports = app;
```

附C5

前端“实时数据”页面功能代码 (page1A.js)

```
$(function(){
init1(1);
$('#loading').modal('show');
})
var dataPressure = new Array();
var dataTempIn = new Array();
var dataTempOut = new Array();
var dataState = new Array();
var fuIdMap = new Array("001","002","003","004","005","006","007");
var presureMap = new Array("Ch6","Ch6","Ch6","Ch6","Ch6","Ch6","Ch6");
var tempInMap = new Array("Ch5","Ch5","Ch5","Ch5","Ch5","Ch5","Ch5");
var tempOutMap1 = new Array("Ch4","Ch4","Ch4","Ch4","Ch4","Ch4","Ch4");
function generateData(jsdata,id) {
var categoryData = [];
var tempInData = [];
var tempOutData = [];
var pressInData = [];
var stateData = [];
var refreshTime = 0;
var refreshState = 0;
```

```
//获取不同时间的数
//console.log('jsd: '+JSON.stringify(jsdata))
var records = jsdata.records;
var rec;
for(var i in records){
rec = records[i];
refreshTime = echarts.format.formatTime('yyyy-MM-dd\nhh:mm:ss', rec.time*1000)
categoryData.push(refreshTime);
tempInData.push(rec.inTemp);
tempOutData.push(rec.outTemp);
pressInData.push(rec.inPress);
refreshState = rec.state
stateData.push(rec.state);
}
dataPressure[id] = {
categoryData : categoryData,
valueData : pressInData
};
dataTempIn[id] = {
categoryData : categoryData,
valueData : tempInData
};
dataTempOut[id] = {
categoryData : categoryData,
valueData : tempOutData
}
dataState[id] = {
categoryData : categoryData,
valueData : stateData,
refreshTime : refreshTime,
refreshState : refreshState
}
console.log(dataState[id].valueData.length)
}
function resolveTime(timeStemp){
var year = timeStemp.substr(0,4);
var month = timeStemp.substr(4,2);
var day = timeStemp.substr(6,2);
var Tindex = timeStemp.indexOf('T');
var hour = timeStemp.substr(Tindex+1,2);
var min = timeStemp.substr(Tindex+3,2);
var sec = timeStemp.substr(Tindex+5,2);
now = new Date(year,month-1,day,hour,min,sec);
return now.setHours(now.getHours()+8);
}
function init1(i){
fetch("/zyf/datafu?FuId="+i, {
method: "GET"
}).then(function(res) {
return res.json();
}).then(function (data) {
var p = new Promise(function (resolve, reject) {
generateData(data,i);
init2(i);
if(i<7){
resolve(i);
}else{
reject();
}
});
p.then(function (i) {
```

```
    init1(i+1);
    }).catch(function (error) {
    console.log('catch error');
    console.log(error);
    });
    });
    }
    function init2(i){
    $('#loading').modal('hide');
    console.log('init2 :' + i);
    var lcs = echarts.init(document.getElementById('lineChart'+i));
    $('#name' + i).text(i+"号釜-----更新时间："+dataState[i].refreshTime+'---状态
:'+dataState[i].refreshState);
    lcs.setOption({
    color:["#87cefa","#ff7f50","#32cd32","#da70d6",],
    toolbox: {
    show: true,
    feature: {
    dataView: {readOnly: false},
    restore: {},
    brush: { type: ['lineX', 'clear']},
    saveAsImage: {}
    },
    },
    tooltip: {
    trigger: 'axis',
    axisPointer: { type: 'shadow' }
    },
    legend: {
    data:['设备内压力','设备内温度','设备表面温度',],
    y: 'top',
    x:'center',
    textStyle:{ color:'#000000', fontSize:20 },
    itemGap : 50
    },
    brush: {
    xAxisIndex: 'all',
    brushLink: 'all',
    brushmode:'multiple',
    throttleDelay:1,
    outOfBrush: {colorAlpha: 1},
    inBrush: { colorAlpha: 1}
    },
    xAxis: {
    data: dataPressure[i].categoryData,
    axisLine:{lineStyle:{color: '#000000'},},
    splitLine: {show: false},
    axisLabel: {textStyle: {color: '#000000',fontSize:18},},
    },
    yAxis :
    [
    {
    type : 'value',
    scale : true,
    axisLine:{
    lineStyle:{
    color: '#000000'
    },
    },
    splitLine: {
    "show": false
```

```
},
axisLabel: {
textStyle: {
color: '#000000',
fontSize:18
},
formatter: function (value) {
return value + "Mpa"
},
},
},
{
type : 'value',
axisLine:{
lineStyle:{
color: '#000000'
},
},
splitLine: {
"show": false
},
axisLabel: {
textStyle: {
color: '#000000',
fontSize:18
},
formatter: function (value) {
return value + "C"
},
},
}
],
dataZoom: [
{ // 这个dataZoom组件，也控制x轴。
type: 'inside', // 这个 dataZoom 组件是 inside 型 dataZoom 组件
start: 70, // 左边在 70% 的位置。
end: 100, // 右边在 60% 的位置。
show: true,
realtime: true
}
],
series : [
{
name:'釜表温度',
type:'line',
itemStyle: {normal: {areaStyle: {type: 'default'}}},
sampling:'average',
smooth:'false',
yAxisIndex: 1,
data:dataState[i].valueData
},
{
name:'釜内温度',
type:'line',
itemStyle: {normal: {areaStyle: {type: 'default'}}},
sampling:'average',
smooth:'false',
yAxisIndex: 1,
data:dataTempIn[i].valueData,
},
{
```

```
name:'釜内压力',
type:'line',
itemStyle: {normal: {areaStyle: {type: 'default'}}},
sampling:'average',
smooth:'false',
data:dataPressure[i].valueData,
}
],
});
}
```

附C6

前端“生产记录”页面功能代码 (page2A.js)

```
$(function(){
init();
})
var zeroOffset = 7;
var offsetDate = new Date();
offsetDate.setTime(offsetDate.getTime()-zeroOffset*60*60*1000);
function addNew(rec){
var FuId = rec.fuId;
var Index = rec.index;
var StartTime = parseInt(rec.startTime);
var EndTime = parseInt(rec.endTime);
var ban = banTime(parseInt(rec.startTime));
var prefix = rec.prefix;
var tb = document.getElementById('fuListTable');
var evStart = timeString(new Date(StartTime),0);
var evEnd = timeString(new Date(EndTime),new Date(StartTime));
var evTotal = diffTime(EndTime,StartTime);
//添加行
var newTR = tb.insertRow(tb.rows.length);
newTR.id = "HH";
//添加列:设备号
var newFuId = newTR.insertCell(0);
//添加列内容
newFuId.innerHTML = FuId+'号釜';
//添加列:序号
var newXuId = newTR.insertCell(1);
//添加列内容
newXuId.innerHTML = Index;
//添加列:开始时间
var newInTime = newTR.insertCell(2);
//添加列内容
newInTime.innerHTML = evStart;
//添加列:出厂时间
var newOutTime = newTR.insertCell(3);
//添加列内容
newOutTime.innerHTML = evEnd;
//添加列:时长
var newTotalTime = newTR.insertCell(4);
//添加列内容
newTotalTime.innerHTML = evTotal;
//添加列:察看按钮
var newMore = newTR.insertCell(5);
//添加列内容
newMore.innerHTML = "<a class='btn btn-lg btn-success'
href='zyrc/zyev?id="+FuId+"&startTime="+StartTime+"&endTime="+EndTime+"&prefix="+prefix+"' target='_Blank' >查
看</a>";
}
function dateString(now){
var year = now.getFullYear();
```

```
var month = now.getMonth()+1;
var date = now.getDate();
var yearStr = year;
var monthStr;
var dateStr;
if(month.toString().length==1){
monthStr = '0'+month.toString();
}else{
monthStr = month.toString();
}
if(date.toString().length==1){
dateStr = '0'+date.toString();
}else{
dateStr = date.toString();
}
var todayStr = yearStr+'-'+monthStr+'-'+dateStr;
return todayStr;
}
function timeString(now,start){
var month = now.getMonth()+1;
var hour = now.getHours();
var date = now.getDate();
var timeStr;
if(month>0){
var min = now.getMinutes();
var minStr;
if(min.toString().length==1){
minStr = '0'+min.toString();
}else{
minStr = min.toString();
}
if(hour>=0 && hour<=6){
return month+'月'+date+'号凌晨'+hour+'点'+minStr+'分';
}else if(hour>6 && hour<13){
return month+'月'+date+'号上午'+hour+'点'+minStr+'分';
}else if(hour>=13 && hour<=18){
return month+'月'+date+'号下午'+(hour-12)+'点'+minStr+'分';
}else{
return month+'月'+date+'号晚上'+(hour-12)+'点'+minStr+'分';
}
}else{
return '等待出釜';
}
}
function banTime(startTime){
var banOffset = new Date(startTime-zeroOffset*60*60*1000);
var startDateStr = banOffset.getMonth()+1+'月'+banOffset.getDate()+'号';
var ban;
if(new Date(startTime).getHours()>=zeroOffset && new Date(startTime).getHours()<12+zeroOffset){
ban = '白班';
}else{
ban = '夜班';
}
return startDateStr+ban;
}
function diffTime(EndTime,StartTime){
if(!(EndTime>0)){
EndTime = new Date().getTime();
}
var time = EndTime-StartTime
var hour = Math.floor(time/1000/3600);
```

```
var min = Math.floor((time-hour*1000*3600)/1000/60);
if(hour<24){
return hour+'小时'+min+'分钟';
}else{
return '见次日记录';
}
}
function showTableData(data){
var tb = document.getElementById('fuListTable');
var rowNum=tb.rows.length;
for (i=1;i<rowNum;i++){
tb.deleteRow(i);
rowNum=rowNum-1;
i=i-1;
}
var outNum = 0;
var outTimeTotal = 0;
var rec;
var evEnd;
for (var i in data) {
rec = data[i];
evEnd = timeString(new Date(parseInt(rec.endTime)),new Date(parseInt(rec.startTime)));
if(evEnd.length>6){
outNum = outNum+1;
outTimeTotal = outTimeTotal+parseInt(rec.endTime)-parseInt(rec.startTime);
}
addNew(rec);
}
var outTotal = outNum*15*4.32;
//$('#text1').text("累计产量："+outTotal.toFixed(2)+"立方");
//$('#text2').text("平均用时："+diffTime(parseInt(outTimeTotal/outNum),0));
}
function init(){
fetch("/zyrc/list?
```

6. 工业物联网数据管理信息系统与终端硬件设计_第6部分

总字数：7540

相似文献列表

去除本人文献复制比：0%(0)　　文字复制比：0%(0)　　疑似剽窃观点：(0)

原文内容

```
date="+dateString(offsetDate), {
method: "GET"
}).then(function(res) {
return res.json();
}).then(function (data) {
showTableData(data);
});
$('.form_date').datetimepicker({
language: 'zh-CN',
weekStart: 1,
todayBtn: 1,
autoclose: 1,
todayHighlight: 1,
startView: 2,
minView: 2,
forceParse: 0
}).on('changeDate', function(event) {
event.preventDefault();
event.stopPropagation();
var day = $('#dtp_input2').val();
```

```
console.log(day);
fetch("/zyrc/list?date="+day, {
method: "GET"
}).then(function(res) {
return res.json();
}).then(function (data) {
showTableData(data);
});
});
}
```

附C7

前端“记录内容”页面功能代码 (page2B.js)

```
$(function () {
init1();
})
var dataPressure;
var dataTempIn;
var dataTempOut;
var dataState;
var zeroOffset = 7;
var offsetDate = new Date();
offsetDate.setTime(offsetDate.getTime() - zeroOffset * 60 * 60 * 1000);
function dateString(now) {
var year = now.getFullYear();
var month = now.getMonth() + 1;
var date = now.getDate();
var yearStr = year;
var monthStr;
var dateStr;
if (month.toString().length == 1) {
monthStr = '0' + month.toString();
} else {
monthStr = month.toString();
}
if (date.toString().length == 1) {
dateStr = '0' + date.toString();
} else {
dateStr = date.toString();
}
var todayStr = yearStr + '-' + monthStr + '-' + dateStr;
return todayStr;
}
function timeString(now, start) {
var month = now.getMonth() + 1;
var hour = now.getHours();
var date = now.getDate();
if (month > 0) {
var min = now.getMinutes();
var minStr;
if (min.toString().length == 1) {
minStr = '0' + min.toString();
} else {
minStr = min.toString();
}
if (hour >= 0 && hour <= 6) {
return month + '月' + date + '号凌晨' + hour + '点' + minStr + '分';
} else if (hour > 6 && hour < 13) {
return month + '月' + date + '号上午' + hour + '点' + minStr + '分';
} else if (hour >= 13 && hour <= 18) {
return month + '月' + date + '号下午' + (hour - 12) + '点' + minStr + '分';
} else {
```

```
return month + '月' + date + '号晚上' + (hour - 12) + '点' + minStr + '分';
}
} else {
return '等待完成';
}
}
function diffTime(EndTime, StartTime) {
if (!(EndTime > 0)) {
EndTime = new Date().getTime();
}
var time = (EndTime - StartTime) / 1000
var hour = Math.floor(time / 3600);
var min = Math.floor((time - hour * 3600) / 60);
return hour + '小时' + min + '分钟';
}
function generateData(jsdata) {
var categoryData = [];
var valueData = [];
var markLine = [];
var rec;
for (var i in jsdata) {
rec = jsdata[i];
categoryData.push(echarts.format.formatTime('yyyy-MM-dd\nhh:mm:ss', new Date(parseInt(rec.t) * 1000)));
valueData.push(rec.v);
}
return {
categoryData: categoryData,
valueData: valueData
};
}
function banTime(startTime) {
var banOffset = new Date(startTime - zeroOffset * 60 * 60 * 1000);
var startDateStr = banOffset.getMonth() + 1 + '月' + banOffset.getDate() + '号';
var ban;
if (new Date(startTime).getHours() > zeroOffset && new Date(startTime).getHours() < 12 + zeroOffset) {
ban = '白班';
} else {
ban = '夜班';
}
return startDateStr + ban;
}
function init1() {
var dateCtrl = 0;
var now = new Date();
now.setTime(now.getTime() + dateCtrl * 24 * 60 * 60 * 1000);
console.log(dateString(now));
fetch("/zyrc/zyevId", {
method: "GET"
}).then(function (res) {
return res.json();
}).then(function (data) {
$('#name_head').text(data.FuId + "号设备");
var startTimeStr = timeString(new Date(parseInt(data.startTime)), 0);
var endTimeStr = timeString(new Date(parseInt(data.endTime)), new Date(parseInt(data.startTime)));
var diffTimeStr = diffTime(parseInt(data.endTime), parseInt(data.startTime));
$('#text1').text("记录开始时间: " + startTimeStr);
if (endTimeStr.length > 6) {
$('#text2').text("记录结束时间: " + endTimeStr);
} else {
$('#text2').text(endTimeStr);
}
```

```
$('#text3').text("记录时长：" + diffTimeStr);
$('#text4').text("采样点数：2403");
$('#text5').text("终端输入：正常");
$('#text6').text("传感器输入：正常");
var recordData = data.data;
dataPressure = generateData(recordData.pressure);
dataTempIn = generateData(recordData.tempIn);
dataTempOut = generateData(recordData.tempOut);
dataState = generateData(recordData.state);
init2();
});
}
function init2() {
var lcs = echarts.init(document.getElementById('lineChart1'));
lcs.setOption({
toolbox: {
show: true,
feature: {
dataView: {readOnly: false},
restore: {},
brush: {
type: ['lineX', 'clear']
},
saveAsImage: {}
},
},
tooltip: {
trigger: 'axis',
axisPointer: {
type: 'shadow'
}
},
legend: {
data: ['设备压力', '设备内温度', '设备表面温度', '阶段阶梯曲线'],
y: 'top',
x: 'center',
textStyle: {
color: '#000000',
fontSize: 20
},
itemGap: 50
},
brush: {
xAxisIndex: 'all',
brushLink: 'all',
brushmode: 'multiple',
throttleDelay: 1,
outOfBrush: {
colorAlpha: 1
},
inBrush: {
colorAlpha: 1
}
},
xAxis: {
data: dataPressure.categoryData,
axisLine: {
lineStyle: {
color: '#000000'
},
},
```

```
splitLine: {
show: false
},
axisLabel: {
textStyle: {
color: '#000000',
fontSize: 18
},
},
},
yAxis: [
{
type: 'value',
scale: true,
axisLine: {
lineStyle: {
color: '#000000'
},
},
splitLine: {
"show": false
},
axisLabel: {
textStyle: {
color: '#000000',
fontSize: 18
},
formatter: function (value) {
return value + "Mpa"
},
},
},
{
type: 'value',
axisLine: {
lineStyle: {
color: '#000000'
},
},
splitLine: {
"show": false
},
axisLabel: {
textStyle: {
color: '#000000',
fontSize: 18
},
formatter: function (value) {
return value + "C"
},
},
}
],
dataZoom: [
{ // 这个dataZoom组件，也控制x轴。
type: 'inside', // 这个 dataZoom 组件是 inside 型 dataZoom 组件
start: 0, // 左边在 10% 的位置。
end: 100 // 右边在 60% 的位置。
}
],
series: [
```

```
{
name: '设备表面温度',
color: 'rgba(27,67,171,0.69)',
type: 'line',
itemStyle: {normal: {areaStyle: {type: 'default'}}},
sampling: 'average',
smooth: 'false',
yAxisIndex: 1,
data: dataTempOut.valueData
},
{
name: '设备内温度',
color: 'rgb(219,186,0)',
type: 'line',
itemStyle: {normal: {areaStyle: {type: 'default'}}},
sampling: 'average',
smooth: 'false',
yAxisIndex: 1,
data: dataTempIn.valueData,
},
{
name: '设备压力',
color: 'rgba(248,36,22,0.69)',
type: 'line',
itemStyle: {normal: {areaStyle: {type: 'default'}}},
sampling: 'average',
smooth: 'false',
data: dataPressure.valueData,
},
{
name: '阶段阶梯曲线',
color: 'rgba(220,216,215,0.69)',
type: 'line',
itemStyle: {normal: {areaStyle: {type: 'default'}}},
sampling: 'average',
smooth: 'false',
show: false,
data: dataState.valueData,
}
],
});
var pcs = echarts.init(document.getElementById('pieChart'));
pcs.setOption({
tooltip: {
trigger: 'item',
formatter: '{a} <br/>{b}: {c} ({d}%)'
},
legend: {
orient: 'vertical',
left: 10,
},
series: [
{
name: '访问来源',
type: 'pie',
selectedMode: 'single',
radius: [0, '30%'],
label: {
position: 'inner'
},
labelLine: {
```

```
show: false
},
data: [
{value: 335, name: '升压', selected: true},
{value: 679, name: '降压'},
{value: 1548, name: '恒压'}
]
},
{
backgroundColor: '#eee',
borderColor: '#aaa',
borderWidth: 1,
borderRadius: 4,
// shadowBlur:3,
// shadowOffsetX: 2,
// shadowOffsetY: 2,
// shadowColor: '#999',
// padding: [0, 7],
rich: {
a: {
color: '#999',
lineHeight: 22,
align: 'center'
},
// abg: {
// backgroundColor: '#333',
// width: '100%',
// align: 'right',
// height: 22,
// borderRadius: [4, 4, 0, 0]
// },
hr: {
borderColor: '#aaa',
width: '100%',
borderWidth: 0.5,
height: 0
},
b: {
fontSize: 16,
lineHeight: 33
},
per: {
color: '#eee',
backgroundColor: '#334455',
padding: [2, 4],
borderRadius: 2
}
}
},
]
}
]
}
);
}
```

URL	目标页面	请求描述
/index	index	返回主页
/page1A	page1A	进入设备参数页面
/page2A	page2A	进入生产记录页面
/page1A/data	--	请求实时数据json文件
/page2A/event	page1B	进入记录内容界面
/page2A/evData	--	请求生产记录列表json文件

/page2A/evAnalysis	--	请求记录分析结果json文件

说明：1.总文字复制比：被检测论文总重合字数在总字数中所占的比例

2.去除引用文献复制比：去除系统识别为引用的文献后，计算出来的重合字数在总字数中所占的比例

3.去除本人文献复制比：去除作者本人文献后，计算出来的重合字数在总字数中所占的比例

4.单篇最大文字复制比：被检测文献与所有相似文献比对后，重合字数占总字数的比例最大的那一篇文献的文字复制比

5.指标是由系统根据《学术论文不端行为的界定标准》自动生成的

6.红色文字表示文字复制部分;绿色文字表示引用部分;棕灰色文字表示作者本人文献部分

7.本报告单仅对您所选择比对资源范围内检测结果负责

amlc@cnki.net

http://check.cnki.net/

http://e.weibo.com/u/3194559873/