

福建工程学院

Fujian University of Technology

毕业设计

题目： 基于微服务架构的云端
数据监控系统设计与实现

学 生： 高 阳

指导老师： 余 景 华

系 别： 电子信息与通信工程系

专 业： 物联网工程

班 级： 物联网 1601

学 号： 3168908107

2020 年 4 月

目 录

摘 要	I
Abstract	II
第 1 章 绪论	1
1.1 引言	1
1.2 选题背景	1
1.3 设计内容	1
1.4 设计的目的及其意义	1
1.5 什么是微服务	1
1.6 前后端分离的开发模式	2
1.6.1 前端设计目标	2
1.6.2 后端设计目标	3
第 2 章 系统总体设计	5
2.1 系统需求分析	5
2.1.1 用户需求	5
2.1.2 设备需求	5
2.2 可行性分析与技术选型	5
2.2.1 前端技术栈	6
2.2.2 后端技术栈	6
2.3 系统架构图	8
2.4 开发和生产环境配置	8
第 3 章 系统详细设计	10
3.1 Cloud Firestore 数据库设计	10
3.1.1 概念设计	10
3.1.2 逻辑设计	10
3.2 用户认证服务的设计与实现	11
3.2.1 注册流程相关的 Cloud Functions	12
3.2.2 登录流程相关的 Cloud Functions	12
3.2.3 更新用户信息相关的 Cloud Functions	12
3.2.4 用户信息认证中间件	12

3.3 设备管理服务的设计与实现	12
3.3.1 添加设备相关的 Cloud Functions	13
3.3.2 编辑设备相关的 Cloud Functions	13
3.3.3 删除设备相关的 Cloud Functions	13
3.3.4 设备查询相关的 Cloud Functions	13
3.4 数据采集服务设计与实现	13
3.4.1 采集数据写入相关的 Cloud Functions	14
3.5 数据可视化的设计与实现	14
第 4 章 系统测试与上云部署	14
4.1 测试说明	15
4.2 上云部署	15
4.2 网络测试	16
总结	18
参考文献	19
致谢	20

福建工程学院本科毕业设计（论文）作者承诺保证书


本人郑重承诺：本篇毕业设计（论文）的内容真实、可靠。如果存在弄虚作假、抄袭的情况，本人愿承担全部责任。

学生签名：

2020年 5月 24日

福建工程学院本科毕业设计（论文）指导教师承诺保证书

本人郑重承诺：我已按有关规定对本篇毕业设计（论文）的选题与内容进行了指导和审核，且提交的毕业设计（论文）终稿与上传至“大学生论文管理系统”检测的电子文档相吻合，未发现弄虚作假、抄袭的现象，本人愿承担指导教师的相关责任。

指导教师签名：

2020年 5月 24日

基于微服务架构的云端数据监控系统设计与实现

摘 要

传统软件设计采用单体应用架构，整个软件应用程序被部署为一个单元，每次功能的变更或缺陷的修复都会需要重新部署整个应用。这种设计方式使得应用随着需求不断的增加、功能不断的完善在单次部署的耗时长，一旦某个地方出现错误会导致整个应用程序无法正常提供服务。特别地，对于可靠性要求高的应用这样的副作用无疑是影响这个业务和体验的。同时，随着客户端计算力的提高，我们可以对应用进行前后端分离，前端专门负责渲染用户界面，而后端来提供数据持久化和各种服务。通过对其部署在不同的服务器上，当后端无法提供服务时，前端依旧有良好的用户体验。然而服务的可靠性依旧很差。于是我们又对后端服务进行分离，将每个领域业务对应的服务拆开单独做成一个服务单独部署，再通过 API 网关进行统一的调度管理，这样就可以保证在其中一个服务宕机的情况下其它服务是正常运行并且受到的影响最小，省时又省力。这就是微服务架构的理念。随着容器技术的出现，微服务架构真正的得以实现。特别是无服务器概念的提出，后端即服务和函数即服务使得微服务设计更容易的得以实现。

在物联网软件架构体系中，常见的软件需求是管理海量的、分布在不同的地理位置的、不同功能类型的物联网设备。这些设备需要在采集数据后送往网关处理中心，其中大部分数据要求实时性且要持久的采集。如果按照传统软件体系设计，当软件体系中的某个服务需要变动或者添加、删除服务时，整个服务会被关闭，然后等待重新编译上线。这会影响到数据的实时性和可靠性，显然这种设计是不再适合这些场景的物联网软件的。本文旨在使用前后端分离，微服务的手段设计并实现物联网软件体系中的数据监控系统，并实施上云部署。

关键词：微服务；前后端分离；数据监控；后端即服务；函数即服务

Design and Implementation of Cloud Data Monitoring System Based on Microservice Architecture

Abstract

Traditional web application architecture means that the entire software program is deployed as a unit and the entire unit must be restarted whenever it needs to be updated. This architecture will reduce the reliability of the entire software and affect the business and user experience. With the improvement of the computing power of the client, we can adapt the method of separating the front end and the back end. The backend provides various business services and data persistence through APIs. The front end mainly calls the API and renders the user interface. After the decoupling of the front and back ends will have a better user experience. But the application is still unreliable. Therefore, we use the service corresponding to each business logic to split the backend into multiple different microservices and deploy separately. The unified scheduling management through the API gateway can guarantee the reliability of the service. In recent years, with the development of cloud computing and container technology, microservices have been implemented. At the same time, serverless hosting makes back-end as a service and function as a service make microservices easier.

A very common situation in the Internet of Things is that sensors send data to the gateway for data analysis by collecting data. So, this has high reliability and real-time requirements for software design. In this paper, this data monitoring application software is designed and implemented by adopting a microservice architecture and deployed to the cloud.

Key words: microservice; front-end back-end separation; data monitoring; back-end as a service; function as a service

第 1 章 绪论

1.1 引言

随着移动通信的发展，特别是 5G 技术逐渐成熟，我们可以将嵌入式设备集成到各种日常用品用以连接互联网实现万物互联互通，打通人与流程、物品之间的通信。如今，越来越多的公司、部门借助低成本的计算、云服务、大数据、统计分析和移动领域等各种技术手段让物品自动的收集、分析和共享数据，最大化的减少人力的消耗来提高生产效率和服务质量。在这样的物联网环境下，数字系统可以按照预先设定的流程记录、监控和调度各个物品。在这之前实施数据监控是必须的。然而，物联网带来的数据量是巨大的，又是极具价值的，但大部分监控体系是单机的、高耦合的，数据本身的产生速度也远远大于以往。数据的频繁读取和写入性能不仅仅是硬件层面的要求，网络承载力和软件并发性也是非常重要。同时许多场景要求低延时，实时性差的数据会导致分析失误或丧失监控的意义。

本文按照物联网软件工程的方法设计了一套基于云端微服务架构的数据（例如温湿度）监控软件系统。旨在借助先进的思想和新兴技术，搭建出符合微服务架构的物联网软件系统来改善单体设计系统的可靠性、扩展性，同时经济易于个人、组织部署使用。

1.2 选题背景

受本科阶段的无线传感网课程设计启发，结合自己工程实践阶段学习到的计算机知识和技能对其优化重构。

1.3 设计内容

(1) 构建微服务并协同多个不同服务，解耦、细化 API 接口来提高扩展性和可靠性，在保证基础后端服务的功能上，又可扩展不同需求。既保证服务的具有透明性，又使得用户体验良好；

(2) 构建易于使用的前端应用，提供可视化的实时数据预览、各种参数设置等。

1.4 设计的目的及其意义

数据监控系统是物联网最常见的系统之一，它在很多场景下有实际应用。例如智能家居的设备状态的监控、学校实验室环境监控、植物和室内环境指数的监控等，方便、高效的监控系统可以让我们更好的对监控对象做出分析改善。

1.5 什么是微服务

首先要明确微服务的规模是视具体的设计有所不同。微服务通常是指通过 HTTP 调用将数十个或数百个微型应用程序连接在一起。因此，微服务架构并不是一个用于处理身份验证、付款和向客户发送电子邮件的整体，而是具有一个用于身份验证，另一个用于支付，另一个用于用营销电子邮件填充收件箱的应用程序，只是服务的颗粒度相较于一个整体较小。

但是，微服务并非没有缺点。在应用拆分之后，每个服务需要单独的部署，这意味着运维人员可能要维护多个服务器，并且保证每个服务器之间的数据是同步。特别随着越来越多的服务加入，要如何治理这些服务的权限、日志分析等问题也都将随之而来。

有没有更好的解决微服务基础设施管理的方案呢？我们可以发现，现在社区是流行于使用 Docker 容器虚拟化技术结合 Kubernetes 容器编排技术做私有部署方案。然而对于个人和某些非盈利组织而言，这样的方案还是没有解决需要维护服务器的措施，并且对于服务的监控、熔断、注册、发现等都是一大难以解决的工程问题。但是随着 Serverless 无服务器概念的提出，我们可以在不关注服务器资源分配的运维问题的情况下就构建出性能卓越的软件应用。针对什么是 Serverless 无服务器概念将在后面的后端设计目标中加以解释。

1.6 前后端分离的开发模式

数据监控系统的目标是构建成跨平台的响应式应用。由于 Web 应用天生就有跨平台的优势，无论是 iOS、MacOS、Android、Windows、Linux 等众多平台均支持 Web 应用，所以本系统将会设计成用户体验接近原生应用体验的 Web 应用。

传统的 Web 应用开发模式是单体应用开发，整个项目的源代码都是放在同一个环境下并且一起部署。视图界面由浏览器客户端发起响应之后通过服务器端模板引擎组装视图结构再返回浏览器客户端进行视图渲染。这使得整个应用是一体的，在项目的初期，单体应用可以很好地运行。然而，随着需求的不断增加，应用的体积会变大，可维护性、灵活性都会下降，特别多人协作比较困难。同时部署前需要长时间的编译、测试，部署时还需要重新启动整个应用，会影响整个应用的运行。

随着芯片行业的发展，计算机设备的计算能力的提高，我们可以将模板引擎到渲染视图部分的工作分配到客户端进行异步更新。这样就可以使得代码库分离，可以有专门的人负责前端和后端的开发和运维。同时对于服务器资源的节省，还可带来更大服务承载力。

1.6.1 前端设计目标

为了让用户对系统的感知更为类似原生应用，前端将构建成渐进式的单页应用。

(1) PWA 渐进式 Web 应用

PWA (Progressive Web Apps, 渐进式 Web 应用) 是使用了现代的 Web API 通过离线缓存等方式来构建 Web 应用的。这种应用的内容可以被 SEO 搜索引擎收录, 可以固定在设备的主屏幕上, 可以简单向网页一样分享它, 可以借助本地数据库离线工作, 跨平台且受到安全策略保护的。它可以获得我们像使用原生应用一样的体验。

目前 PWA 应用已经受到了浏览器的广泛支持, 包括 Google Chrome 浏览器、Firefox 火狐浏览器、Microsoft Edge (Edge HTML) 浏览器、Microsoft Edge (Based Chromium) 浏览器和 Safari 浏览器。

(2) SPA 单页应用

SPA (Single-Page Application, 单页应用) 与传统的应用从服务器模板渲染引擎渲染不同, 它是过动态的重写当前页面的 DOM 来实现用户交互的。这种方法避免在页面之间来回切换时打断用户体验, 使应用程序更像一个桌面应用程序。

现代浏览器均支持大部分标准的 HTML5、CSS3 以及 ECMAScript2015 新特性, 行业已经有诸多实践可以供给参考。

1.6.2 后端设计目标

为了构建高可靠、实时性强的数据监控系统, 后端对服务进行微服务解耦, 围绕 FaaS 设计 API 并通过 Serverless 无服务器的云部署架构实现颗粒度小、松耦合的后端服务。

(1) BaaS 后端即服务

BaaS (Back-end as a Service, 后端即服务) 涵盖的范围比较广泛, 包含了应用所依赖的任何服务。它通过提供各种各样的服务来方便集成和使用, 而无须使用者关心如何运维。比如需要使用数据库服务, 你无需下载、安装、配置和部署数据库, 只要向服务中心申请资源, 获得权限拿来即用。

(2) FaaS 函数即服务

FaaS (Function as a Service, 函数即服务) 提供了一个基础的运行环境, 在这个环境中你可以使用一个函数或者多个函数来组合成一个应用提供服务, 一般支持大多数常用的编程语言。FaaS 的计算资源受计算平台视访问量的多少来动态分配。FaaS 常与 BaaS 配合使用, 也可以单独使用。它是受事件驱动 (Event Driven) 的, 可以结合不同的触发器 (trigger) 实现大多数函数逻辑。

(3) Serverless 无服务器部署

为了实现对应用服务的解耦以及合理分配服务器资源, 与传统架构相比, Serverless 架构有如下的几个特点:

①事件驱动: 在 Serverless 架构下, 应用加载和执行被设计为事件驱动的, 通过 Webhook 等技术设置不同的触发器和预先编写的函数相互绑定, 然后针对不同事件具有不同的反应动作 (action), 这样就可以做到对事件的整个生命周期进行控制。

②按需加载：应用不能保持长时间的允许，一般平台会设定一个运行时间，超时后就会被强制卸载，无法持续占用计算资源。应用的启动需要一段时间来完成，因此应用具有一个冷启动时间，当然这个时间不会太长。自然地，平台发现某个函数被重复调用的时候，就会对其进行优化。

③状态非本地持久化：由于应用受到平台的资源分配，因此应用的实例不再与任何的服务器资源相绑定，而是分布式的存在于任何空闲的服务器上的。与传统的非解耦设计相比较，Serverless 应用是无法保持本地状态的持久化的。

④非会话保持：由于状态非本地持久化，那么应用每次运行所生成的实例都可能处在不同的服务器资源上，当该应用想进行一个会话保持（Sticky Session）的时候，它无法获得关联的应用上下文（context）。因此，对于长链接有状态的应用是不合适的。

⑤自动弹性伸缩：由于应用是按需加载的，当有突发的并发请求时，云计算服务平台会根据自身的计算资源和应用实例的实际需求动态的调整，在应用闲置的时候及时的回收计算资源，实现弹性的自动伸缩和扩展。

⑥应用函数化：每个业务服务均是由 1 个动作或者多个相关联的动作构成的，因此应用会被解耦成多个颗粒度细小的函数，而每个函数的状态和会话都无法持久化。

第 2 章 系统总体设计

2.1 系统需求分析

基于微服务架构的云端数据监控系统被设计成一个具有实时性高、扩展性强、安全性高的系统，用户可以根据自己想要查看或控制的环境变量需求去增加或删除前端物联网设备，并对其进行属性设置。通过仪表盘能够个性化定制屏幕数据可视化分析的布局结构，在设备列表还能看到设备的状态和其历史采集数据等。

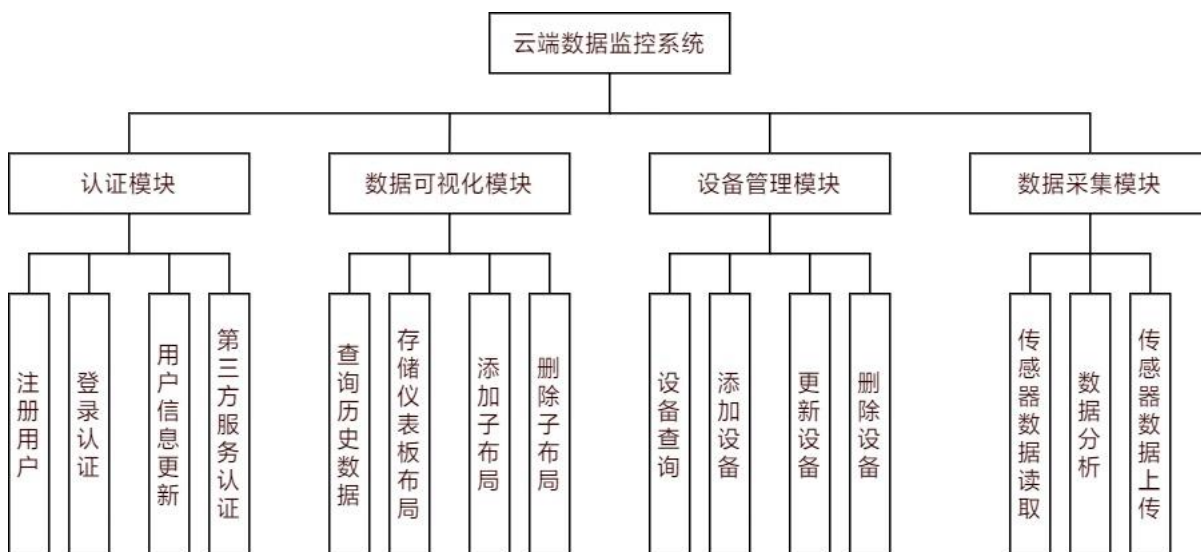


图 2-1 系统总体功能结构图

2.1.1 用户需求

- (1) 为了确保用户设备安全以及考虑部署到公网环境，设计为无游客访问功能；
- (2) 用户可以注册、登录，修改自己的头像、密码、姓名、邮箱等基础信息；
- (3) 用户可以查看、添加、编辑、删除自己的物联网设备；
- (4) 用户可以在仪表盘新增或删除仪表布局，方便查看实时数据。

2.1.2 设备需求

- (1) 设备需要存有历史采集数据；
- (2) 设备对数据库特定文档有写入数据的功能，尽量做到与通信协议无关；
- (3) 设备端可以离线使用，在网络波动恢复后自动同步数据；
- (4) 对于设备采集的数据，要有方法可以拓展数据分析。

2.2 可行性分析与技术选型

Web 应用的构建中，由于 Node.js 和 ECMAScript2015 的出现，JavaScript 被赋予了后端开发的能力。对于个人构建整个程序来说选择 JavaScript 全栈是最理想和快速的。

本数据监控系统全栈使用 JavaScript 构建，结合 React.js 库及其生态构成的前端框架和 Firebase SDK for Node.js 提供的驱动 Firebase 云移动平台能力构建成新型的物联网云端数据监控系统。

2.2.1 前端技术栈

(1) TypeScript

TypeScript 是 Microsoft 维护的一门类型安全的开源编程语言。TypeScript 带来了可选的静态类型检查以及最新的 ECMAScript 特性，是 JavaScript 类型的超集，因此它可以被编译成普通的 JavaScript，从而运行在跨平台系统上。

(2) React.js+React-Router+Material-UI 框架

React.js 是 Facebook 公司开源的用于构建用户界面的 JavaScript 库，是目前最为流行的单项数据流前端库，结合 React-Router 作为路由管理、React-Redux 作为数据状态管理、Material-UI 遵循 Google Material Design 规范响应式 UI 组件库作为用户 UI 界面可以构建出完整的单页渐进式 Web 前端程序，如果需要真正原生的体验结合 Electron.js、React Native 等技术可以将现有的代码转为 iOS、Android 等平台的原生应用程序。本数据监控系统仅仅构建出 PWA 渐进式应用，原生应用技术栈不再赘述。

(3) Recharts 数据可视化 React 组件

Recharts 是一个轻量级基于 D3.js 数据可视化子模块的解耦的、可重用的 React 图表组件。它可以简单、快速的绘制出 SVG 高质量元素，保证在各种分辨率情况下，图表都具有高质量的渲染。数据监控系统仪表板中所有实时数据的可视化均由 Recharts 绘制。

2.2.2 后端技术栈

Firebase 是 Google 面向移动应用和 Web 应用程序开发者的云平台。它集成了包含数据库服务、机器学习服务、身份认证服务、静态文件托管服务、大型媒体文件托管服务、质量分析服务、消息推送服务等移动服务产品，同时结合 Google Cloud 提供 BaaS 和 FaaS，能为开发者提供快速在 Android、iOS、Web、C++、Unity 等平台上构建应用的能力。Firebase 提供了免费的入门方案，包括完整基础服务调用和性能分析，调用时长根据服务不同有不同的额度，体验下来对于个人和小型团队都是完全足够使用的。针对免费套餐额度使用完之后还可以按照使用时长、使用次数等不同维度的计费方案付费使用。这种完全免费全托管的方式无论是对个人开发者还是团队开发都非常经济、快速、可行的。

云端数据监控系统的后端将由 Firebase 驱动，围绕 Cloud Functions 构建微服务。主要使用了 Firebase Authentication 身份认证服务、Cloud Firestore 实时数据库服务、Cloud Storage 文件存储服务、Firebase Hosting 云托管服务、Cloud Functions for Firebase 云函数服务。

(1) Firebase Authentication 身份认证服务

Firebase Authentication 身份认证服务可以轻松与其他 **Firebase** 服务集成，它是按照行业安全标准（**OAuth 2.0** 和 **OpenID Connect** 等）来设计的，因此亦可与自定义后端轻松集成。其包括基于电子邮件和密码的身份验证、联合身份提供方服务集成（例如 **GitHub**）、电话号码身份验证、自定义身份验证系统集成以及匿名身份验证。其工作原理是用户通过获取凭证发送给 **Firebase Authentication SDK** 等待验证后获得对 **Firebase** 产品和自定义后端服务的访问权。这些凭证可以是电子邮件与密码，也可以是联合身份提供者授权的 **OAuth** 令牌。在数据监控系统中，将集成 **Firebase Authentication** 的电子邮件认证服务以及通知服务。

（2）**Cloud Firestore** 实时数据库服务

Cloud Firestore 是一种部署在云端的 **NoSQL** 文档数据库，借助强大的 **Google Cloud** 它可以在全球范围内轻松存储、同步和查询移动应用及 **Web** 应用的数据。并且这些数据可以实时的同步到所有客户端的。区别于普通的数据库，它还有离线的存储功能，在网络不稳定的情况下，数据会在客户端先缓存，等待网络恢复时会自动同步到远程数据库。这具有非常良好的可靠性，非常适合用于物联网系统。因此，在数据监控系统中，将集成 **Cloud Firestore** 提供传感器数据存储和自定义后端服务的数据持久化，同时其跨设备的实时性可以让监控系统比设置定时任务获取最新数据的方式更加具有实时性。

特别地，在 **Cloud Firestore** 中，存储单元是文档。类似 **JSON** 每个文档都用一个 **key** 进行标识，键就是传统意义的唯一标识 **id**。其支持的数据类型的值有：布尔值、数字、字符串、地理坐标点、二进制 **blob** 和时间戳。

（3）**Cloud Storage** 文件存储服务

Cloud Storage 是一种快速安全存储和提供用户文件的内容的托管服务。它会根据用户身份或文件属性（例如文件名、大小、内容类型和其他元数据）来确定是否允许访问。得益于 **Google** 全球 **CDN**，无论在哪里用户都可以快速的获得想要的文件，并在网络状况不好的情况下提供暂停和自动恢复下载的功能，节省用户的带宽和时间，安全可靠。在数据监控系统中，将集成 **Cloud Storage** 服务提供用户头像、文件快照等存储和读取的服务。

（4）**Firebase Hosting** 云托管服务

Firebase Hosting 是专门为现代 **Web** 应用设计的托管服务，无论是部署移动应用的着陆页，还是部署单页 **Web** 应用或渐进式应用都可以通过全球 **CDN** 快速的访问，部署会在极短的时间内应用到全球的 **CDN** 节点服务，同时具有路由重写功能。特别地，它还提供了免费的网域服务，轻松获取 **SSL** 证书，保持整个应用的访问是加密安全的。在数据监控系统中 **Firebase Hosting** 将托管前端生成静态文件。

（5）**Cloud Functions for Firebase** 云函数服务

Cloud Functions for Firebase 提供了托管自定义函数的能力。通过 **HTTP** 函数，可以方便的获取客户端传来的认证凭据和消息服务，然后执行相应的逻辑。同时它还提供了

与其它 Firebase 服务交互的能力，例如通过各种类型的 trigger 进行事件监听，可以在 Cloud Firestore 实时数据库发生改变的时候触发一些算法进行数据分析。这对实时数据流的动态分析是非常方便的，可以轻松知道数据库发生了什么变更，必要的时候可以触发消息通知服务。如果用户监控的数据比较重要，当数据异常波动时可以快速的提醒用户。同时云函数是在有需要的时候才会加载运行，在闲置的时候就会被卸载，可以节省计算资源和带宽资源。每个函数有自己的生命周期，但函数更新的时候，旧的实例会被自动替换为新的实例。在数据监控系统中，将会使用 Cloud Functions 来做自定义的服务，比如数据分析、消息推送和数据持久化等。

2.3 系统架构图

根据上述的系统设计可总结得系统软件架构图如下：a

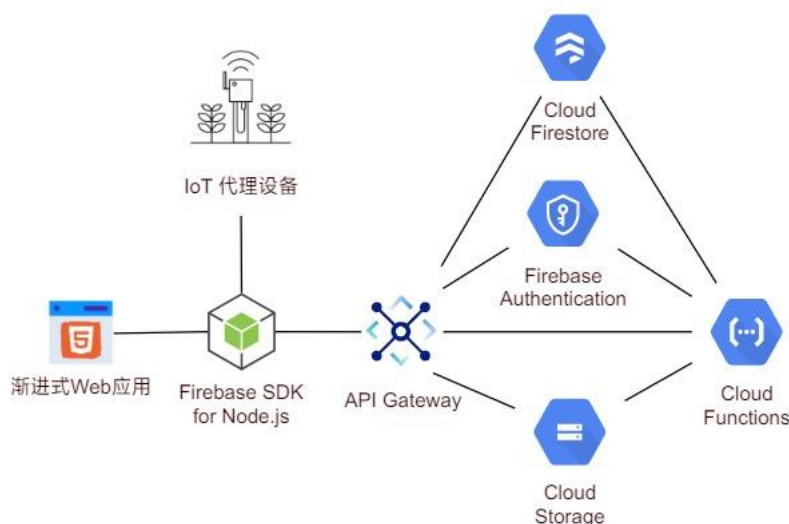


图 2-2 系统结构图

2.4 开发和生产环境配置

- (1) 开发工具：Visual Studio Code。
- (2) 技术栈版本：Node.js 12.0, TypeScript 3.8。
- (3) 环境变量（下述 Firebase 相关的环境变量均在 Firebase 控制台获取）：
 - APP_API_URL 是 API 统一网关的地址；
 - APP_API_KEY 是 Firebase SDK 的密钥；
 - APP_AUTH_DOMAIN 是 Firebase SDK 认证白名单域名地址；
 - APP_DATABASE_URL 是 Cloud Firestore 实时数据库的访问地址；
 - APP_PROJECT_ID 是在 Firebase 控制台中创建的项目名称；
 - APP_STORAGE BuckETS 是 Cloud Storage 存储地址；
 - APP_MESSAGING_SERNDER_ID 是 Firebase 用于消息推送服务 ID；
 - APP_WEB_ID 是 Firebase 的应用标识；

APP_MEASUREMENT_ID 是 Firebase 测量编号。

- (4) Firebase Emulator 本地模拟器，基于 Node.js 10.0。
- (5) Firebase CLI 使用命令行初始化和管理工作，基于 Node.js。
- (6) 调试 React Virtual DOM: React Developer Tools for Chrome。
- (7) 代码版本控制管理: Git 2.26。

第 3 章 系统详细设计

3.1 Cloud Firestore 数据库设计

基于 Cloud Firestore 本身是一种文档型的 NoSQL，其结构类似于 JSON 数据，允许各个集合拥有子集合，子集合可以嵌套子集合，允许嵌套 100 级，目的是为了更方便查询。

3.1.1 概念设计

(1) 用户实体是用来对用户的基本信息进行描述和各种服务认证的，包括了姓名、头像地址、邮箱地址、布局信息存储等的属性。其具有一些与 Firebase Authentication 相冗余的数据，为的是方便用户的管理。

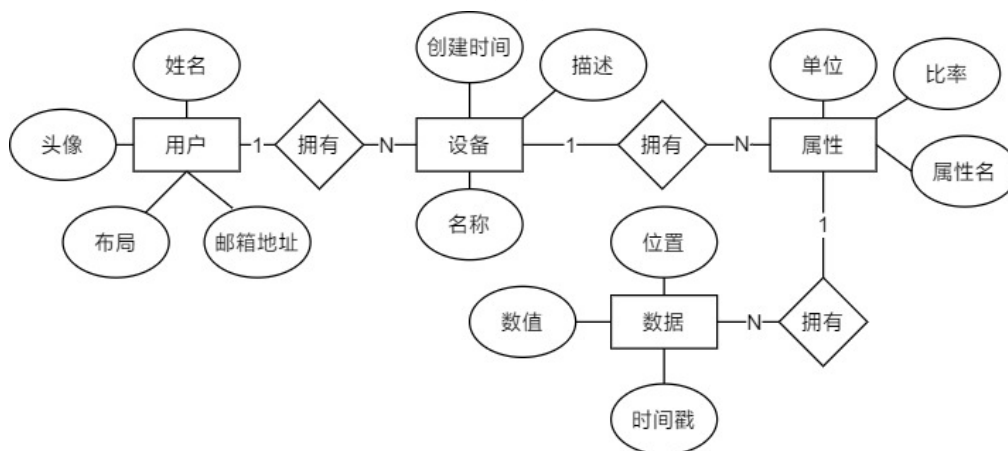
(2) 设备实体是用来对前端物联网数据采集传感器设备的基本特征的描述和身份识别，其包括了每个传感器设备的名称、地理位置、备注描述、创建时间和采集属性等基本属性。

(3) 属性实体是用来对前端物联网数据采集传感器设备所采集的数据的一个描述，其包括了每个采集样本的属性名称、转换单位、转换比率（用于显示）等属性。由于属性的不确定性，所以有单独的实体对其进行描述，以便扩展系统的功能性。比如在前端定义其数据结构，这在前端界面设计中会有体现。

(4) 数据实体是用来对前端物联网数据采集传感器设备所采集的数据的元数据的描述，它包含了所采集数据的负荷载体以及采集的时间戳。

为了方便云端数据监控系统的功能设计，设计有每个用户可以拥有多个设备实体，每个设备可以拥有多个属性实体，每个属性实体可以拥有多个数据实体。数据库实体关系图如下：

图 3-1 数据库实体关系图



3.1.2 逻辑设计

根据数据库实体关系图转化成结构数据后得到如下的数据库文档结构：

(1) 用户 users 的文档结构:

表 3-1 users document

字段名称	字段类型	数据描述
name	string	用户名称
avatar_url	string	头像链接
device_ref	reference	设备集合的路径
email	string	邮箱
layouts	array	布局数据集

(2) 设备 devices 的文档结构:

表 3-2 devices document

字段名称	字段类型	数据描述
name	string	设备名称
description	string	设备描述
user_ref	reference	用户集合的路径
values	sub set	设备属性子集合

(3) 设备 values 的文档结构:

表 3-3 values document

字段名称	字段类型	数据描述
name	string	设备名称
unit	string	属性单位
display_ratio	number	显示比率
data	sub set	属性数据子集合

(4) 属性 data 的文档结构:

表 3-4 data document

字段名称	字段类型	数据描述
payload	number	数值
timestamp	timestamp	采集时间
location	geopoint	采集坐标

3.2 用户认证服务的设计与实现

3.2.1 注册流程相关的 Cloud Functions

(1) 创建用户（基于 POST HTTP 函数）

用户在前端正确的填写表单信息，点击注册按钮之后以 POST 的方式发起 HTTP 请求。通过 Admin SDK 调用 `admin.auth()` 的 `createUser` 方法并传入 `email` 和 `password` 等参数，然后返回响应结果。前端在收到响应后会根据是否成功会跳转自相应的路由。

(2) 冗余用户数据（基于 Trigger 函数）

注册过程中需要对用户 `uid` 进行一个冗余存储，因此通过在 `functions.auth.user()` 上添加 `onCreate` 监听用户创建事件，当有用户创建时就往 `Firestore` 的 `users` 集合中写入冗余信息。

3.2.2 登录流程相关的 Cloud Functions

通过 Email 和密码登录（基于 POST HTTP 函数）：用户在前端正确的填写登录表单信息，点击登录按钮后以 POST 的方式发起 HTTP 请求。通过 Admin SDK 调用 `admin.auth()` 的 `loginWithEmailAndPassword` 方法获得用户信息和凭证，如果成功将凭证设置到响应的请求头信息以便后续的认证，返回用户信息。前端收到响应后根据是否认证通过跳转到相应的路由。

3.2.3 更新用户信息相关的 Cloud Functions

(1) 更新用户的文档资料（基于 PUT HTTP 函数）

用户在前端的填写需要修改的相关信息表单后携带用户 `uid` 以 PUT 的方式发起 HTTP 请求。通过 Admin SDK 调用 `admin.auth()` 的 `updateUser` 方法更新用户信息，获得操作结果换取最新的用户信息后同步更新 `Firestore` 中的 `user` 文档数据。

(2) 上传用户头像（基于 POST HTTP 函数）

用户在前端的点击上传图片的按钮，选择合适的图片后通过前端 `Blob` 编码后以 POST 的方式发起 HTTP 请求。通过 Admin SDK 调用 `Cloud Storage` 的上传文件方法，等待上传完成后返回一个 URL 地址，再将这个地址结果返回。

3.2.4 用户信息认证中间件

验证当前登录状态被设计成一个请求验证的中间件，原理是基于洋葱模型对除注册、登录、公共访问区域以外的所有 `Cloud Function` 的请求头拦截并进行解析，获得相关的登录凭证后与 `Firebase` 认证的凭证相比较，如果有凭证过期或者不全或者伪造再会对响应清除所有的认证相关的响应信息，后续的执行全部中断。如果凭证依旧有效就执行真正的请求函数体。前端根据这个响应做出是否清除本地 `Cookies`、`Local Storage` 和 `Firebase` 退出等操作。

3.3 设备管理服务的设计与实现

3.3.1 添加设备相关的 Cloud Functions

添加设备（基于 POST HTTP 函数）：用户在前端需要对设备的属性进行定义，例如这个设备是采集温度的，那么它的属性中会有一个“温度”的属性，并有其单位的其它附加属性。点击添加设备按钮之后以 POST 的方式携带整个验证后的表单发起 HTTP 请求。收到请求后，通过 Admin SDK 调用 `admin.firestore()` 的相关数据库方法设定该设备的信息，并将其与用户进行关联，然后返回响应结果。

3.3.2 编辑设备相关的 Cloud Functions

编辑设备（基于 PATCH HTTP 函数）：用户在前端只能对除现有数据需要对设备的属性进行定义，例如这个设备是采集温度的，那么它的属性中会有一个“温度”的属性，并有其单位的其它附加属性（不包括所属用户的参考）。点击添加设备按钮之后以 PATCH 的方式携带整个验证后的表单发起 HTTP 请求。收到请求后，通过 Admin SDK 调用 `admin.firestore()` 的相关数据库方法更新该设备的信息，然后返回响应结果。

3.3.3 删除设备相关的 Cloud Functions

删除设备（基于 DELETE HTTP 函数）：由于设备和 Firestore 的特殊性，需要对所有的子集合进行删除，因此对于设备删除操作是用户在前端只能对除现有数据需要对设备的属性进行定义，例如这个我们仅仅是把设备的状态设置为隐藏，这样对于历史数据的保存还是完整的，也不涉及复杂的删除操作。点击删除设备按钮之后以 DELETE 的方式携带设备的 id 发起 HTTP 请求。收到请求后，通过 Admin SDK 调用 `admin.firestore()` 的相关数据库方法设定该设备为隐藏，然后返回响应结果。

3.3.4 设备查询相关的 Cloud Functions

（1）查询设备信息（基于 GET HTTP 函数）

用户在前端点击需要查询的设备之后以 GET 的方式携带设备的 id 发起 HTTP 请求。收到请求后，通过 Admin SDK 调用 `admin.firestore()` 的相关数据库方法获取设备的基本信息，然后返回响应结果。

（2）查询设备数据（基于 GET HTTP 函数）

用户在前端选择需要查询的数据的时间范围，点击查询按钮之后以 GET 的方式携带设备的 id 发起 HTTP 请求。收到请求后，通过 Admin SDK 调用 `admin.firestore()` 的相关数据库方法去筛选时间范围内的设备采集数据，然后返回响应结果。

3.4 数据采集服务设计与实现

由于物联网通信协议还尚未得到统一，MQTT 等常用的协议也有复杂的版本迭代。为了做到扩展性强，数据采集方面采用客户端模拟的方式，提供 API 让用户自行适配。

3.4.1 采集数据写入相关的 Cloud Functions

数据采集（基于 POST HTTP Function）：采集前端通过绑定相应传感器 id 获取到采集时间和数值后以 POST 的方式携带采集数据发起 HTTP 请求。收到请求，通过 Admin SDK 调用 `admin.firestore()` 的相关数据库方法将数据写入到其 `data` 文档中，结束响应。

3.5 数据可视化的设计与实现

由于 Serverless 中，FaaS 目前只适用于无状态的链接，每个函数在冷启动后运行周期不可超过 60s，因此对于实时服务是不合适。对于这个问题的解决方法是直接通过 Firestore 的 SDK 在前端直接访问数据库并监听数据变化。这会在前端建立起一个 WebSocket 的链接保持与数据库的通信。这可能不符合微服务中网关的设计方法，但是对于有状态的服务，一般网关都是永久代理转发这个服务，跟直接访问没有太大区别。

在前端，依托强大的 D3.js，只需要提供数据，我们就可以方便定制各式各样的图表。由于图表的多样，功能不一，现只提供几种常用图表的显示。只需要选择要显示的属性就可以。对于单值显示就和正常文字显示没有区别了。

第 4 章 系统测试与上云部署

4.1 测试说明

整个系统在技术选型的时候就选择了类型安全的 TypeScript 来构建，并且得益于 Visual Studio Code 本身对 TypeScript 的良好 Debug 支持，以及 Visual Studio IntelliCode For Visual Studio Code 使用 AI 辅助功能，通过上下文智能感知，代码样式的推理和执行等功能已经在开发阶段对代码进行了类型安全编译检查和分支推理，已经相当于代码本身的白盒测试。特别地，对于前端是面向用户界面的，对于测试需要比较高的要求，测试要模拟整个 HTML DOM 来进行 UI 测试，因此不再详细测试，而使用 React Developer Tools 进行简单的性能测试。对于后端，BaaS 已经是稳定的 API 接口，经过 Firebase 团队详细的测试。

综上所述，不对整个系统展开详细的测试，而是转而对于网络可靠性带来的影响使用 Chrome DevTools 进行性能测试。

4.2 上云部署

Firebase 云部署非常简单，在初始化项目的时候就已经通过提示在本地写入了相关的配置文件，如下图所示：

```
→ create-react-app-with-typescript git:(master) X firebase init

#####
##
#####
##
##

You're about to initialize a Firebase project in this directory:

C:\Users\redblue\Documents\JustCode\create-react-app-with-typescript

? Are you ready to proceed? Yes
? Which Firebase CLI features do you want to set up for this folder? Press Space to select features, then Enter to confirm your choices.
> ( ) Database: Deploy Firebase Realtime Database Rules
(*) Firestore: Deploy rules and create indexes for Firestore
(*) Functions: Configure and deploy Cloud Functions
(*) Hosting: Configure and deploy Firebase Hosting sites
(*) Storage: Deploy Cloud Storage security rules
(*) Emulators: Set up local emulators for Firebase features

Ln 37, Col 21 Spaces: 2 UTF-8 LF TypeScript React 3.8.3 Prettier: ✓
```

图 4-1 Firebase 初始化

运行 `npm run build` 命令获得编译后的文件，然后使用 `firebase deploy` 命令即可一键部署。部署完成后会获得一个带安全证书的 HTTPS 访问链接：`https://fjut-2020.web.app/`，在 Firebase Console 查看所有部署的内容，例如下图所示：

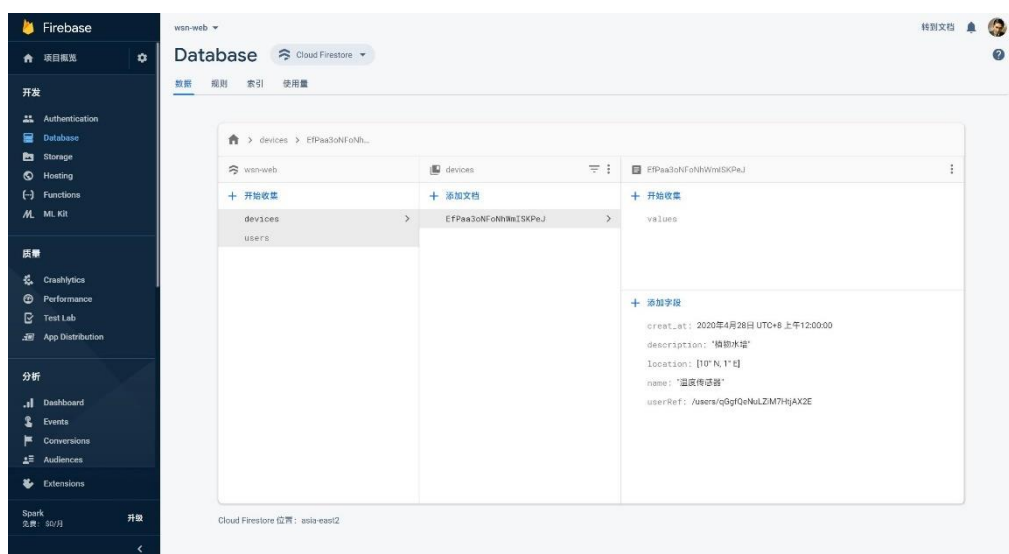


图 4-2 Firebase 控制台

4.2 网络测试

Lighthouse 是一个开源的自动化工具，用于分析和改进网络应用的质量。它集成在 Chrome DevTools 中，运行 Lighthouse 对页面进行一连串测试，生成一个有关在桌面和移动端页面性能的报告，如下图所示，报告指出应用的运行状况在不同的环境下均运行流畅。

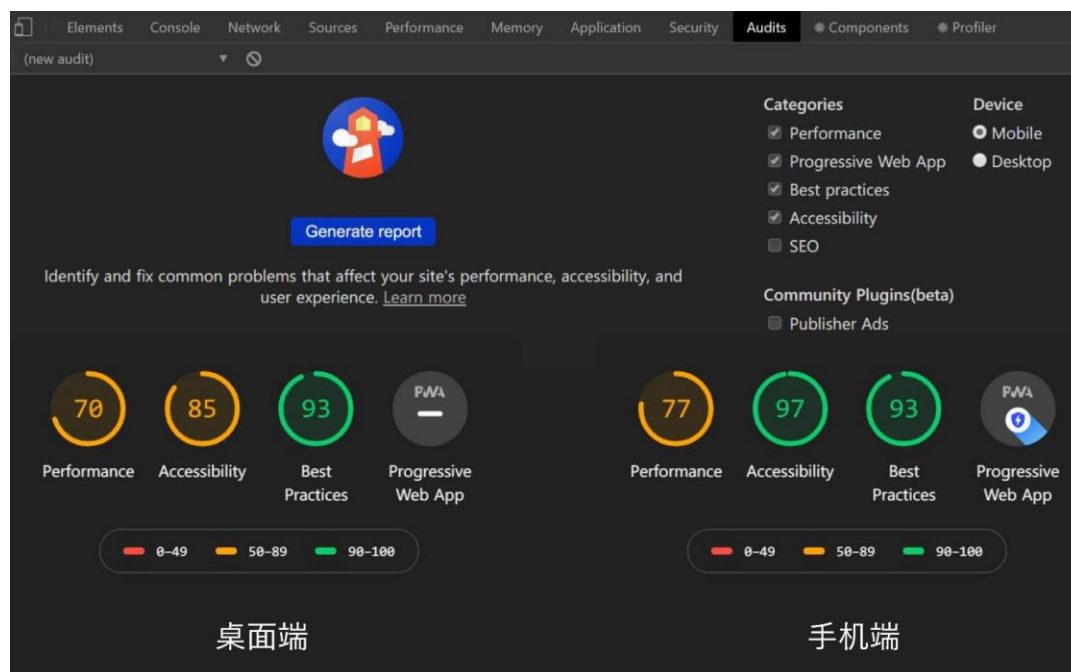


图 4-3 Lighthouse 性能报告

针对后端服务响应测试，简单的几次访问后，可到 **Firebase** 后台查看性能分析。分析显示得益于全球 **CDN** 加速和平台能力，在大部分场景下响应均保持良好，如下图所示：

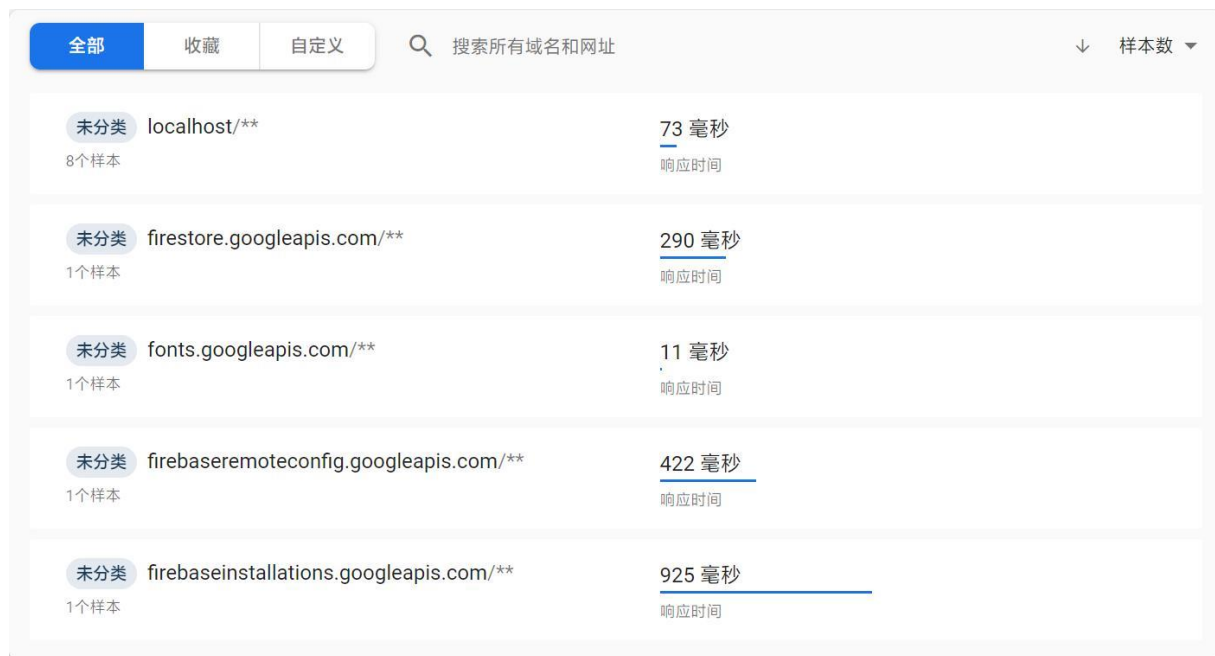


图 4-4 Firebase 性能分析报告

这里应当指出，软件性能可靠性不是靠一时的测试能够看出来的，它需要经过时间和各种场景的考验，这里仅供参考。

总结

这次毕业设计在前期由于对相关领域了解不足够，导致在后期实践中出现多次错误设计或者是过渡的设计，归根结底还是不够有耐心看其核心设计思想和归纳特征。在云端数据监控系统中，前端核心是围绕 React 生态构建，它以 Virtual DOM 和原生的 DOM 进行 Diff 来获得修改 DOM 次数最少的方法，一改 jQuery 时代的直接操作原生 DOM 带来的复杂的性能消耗。一开始对其原理的不理解，还试图使用 jQuery 的思想实践，结果与其原理相悖，造成了更大的性能损失，将优化转成了负优化。而后端是依托 Firebase 云生态构建，在设计微服务部分的时候，对服务的边界划分不够清晰，导致与微服务思想不符。开发过程中难免遇到困难，有的问题可能一直卡住，这时候尽管很难，但是坚持下来总有解决办法。

如今科技发展越来越迅猛，新奇的工具和技术层出不穷，这些工具和技术无疑都是为了提高我们的效率。但是科技的快速迭代，总是会让人有些感觉跟不上。但是通过几个月的工程实践，我发现只要对基础知识有足够的理解，多翻看官方文档，掌握其设计思路和模型，再通过简单的实践，并把他们用在合适的领域，是能够快速的掌握他们并发挥他们的作用的。

总的来说，这个云端数据监控系统从设计到实现是一次新的尝试，无论是架构层面还是技术层面都是比较新和流行的内容。虽然它依靠 Firebase 平台获得了服务可靠、成本低的优点，但是它对平台过于依赖，Firebase 的限制就是它能实现的最大功能。不过好在架构设计上是微服务的，微服务的设计本身就很容易集成第三方服务，与平台无关的。其次对于可视化部分如何动态生成不同的图表也是极具挑战的，目前系统大部分功能还比较单一。不过总体上看，我认为系统基本达到了预期设计的目标。

参考文献

- [1] 迟江波. 基于 Web 的环境监控数据查询系统设计研究[J]. 环境科学与管理,2018(06):138-141
- [2] 李莎莎. 基于 Serverless 的前端开发模式研究[J]. 电脑知识与技术,2019(29):246-247
- [3] 佚名. Serverless 架构在 IoT 领域的应用[J]. 电脑编程技巧与维护,2017(17):6
- [4] 佚名. 物联网开发需要 Serverless 架构[J]. 电脑编程技巧与维护,2016(15):6
- [5] 陈孟茜. 微服务架构在分布式管理系统中的应用与设计[J]. 内江科技,2019(12):11+32
- [6] 高宏宇,王鸿磊,凌启东. 基于 NB-IOT 的云平台无线数据监控系统设计[J]. 河北软件职业技术学院学报,2019(01):16-19
- [7] 赵军,陈子晗,高子航. 基于微服务架构的一体化科研管理平台设计与实现[J]. 无线电工程,2019(05):84-89
- [8] 裴宏祥,于晓虹. 基于微服务架构的系统设计与开发[J]. 中国科技信息,2019(10):94-95
- [9] 周乐钦,燕彩蓉,苏厚勤. 基于 Web-Socket 协议的推送数据技术在监控系统中的应用研究[J]. 计算机应用与软件,2013(05):235-238
- [10] 罗学强,何斌. 基于 Web 的远程数据采集监控系统的应用[J]. 自动化与信息工程,2006(04):25-27
- [11] 张玲,张翠肖. WebSocket 服务器推送技术的研究[J]. 河北省科学院学报,2014(02):53-57
- [12] 赵亮,赵绪辉. 基于 NodeJS 的跨平台聊天应用的设计与实现[J]. 电脑迷,2017(01):34-35
- [13] 柳志强,陕粉丽. 基于 NodeJS 的聊天系统的设计与实现[J]. 电脑知识与技术,2017(13):75-76
- [14] 李锦涛. 基于 Nodejs 全开放式匿名交流平台的搭建[J]. 科技传播,2018(03):169-171
- [15] 崔丹. BaaS 平台:移动互联网与云计算融合的产物[J]. 软件和信息服务,2012(08):11
- [16] Long Sun, Yan Li, Raheel Ahmed Memon. An Open IoT Framework Based on Microservices Architecture[J]. 中国通信,2017(02):159-167
- [17] Mayuri Chandakant Wadkar , Priyanka Patangrao Patil. Traditional Infrastructure vs. Firebase Infrastructure. 2018, 2(4)

致谢

今年毕业季恰逢全球冠状病毒流行期，首先要感谢党和国家的领导，以及全体在战役过程中挺身而出的医护人员、自愿者等前线人员的艰苦付出，我们才能在疫情期间健康、无忧的准备毕业设计。

其次我要感谢我的父母在这二十多年来无微不至的照顾和关怀，给予我物质上和精神上的满足。特别是大学期间这些远离家乡的日子里，他们让我知道，无论我身处何处，他们都在远方挂住我，无论我遭遇怎样的不顺，他们都是最爱我的人。

再有我要特别感谢大学期间余庚老师、陈新元老师、张荣江老师、曾祝明老师、孟圣慧老师、王黎欣老师以及余景华导师不仅授予我知识、给予我学习和生活上的帮助，还以身作则让我明白大学的意义、学习的重要性和教育的目的。终身学习，学无止境。还要感谢田成孝辅导员在大学生活和学习中给予的关心和付出。

随后我要感谢在大学期间遇到的每一个志趣相投、志同道合的人，是你们让我相信努力就会有收获，是你们给予我快乐的大学的生活，是你们在我需要帮助的时候向我伸出手。感谢福建工程学院中世竞创协会、福建工程学院鱗溪易班工作站、福建工程学院网科组、福建工程学院党员学习与实践中心实践/督导部、福建工程学院计算机协会和电子协会，是它们给予我丰富的课余生活和管理经验，让我不荒废大学里除课程学习外的时光。

最后，我要感谢我自己。虽然身处异乡，远离原来的亲朋好友，但是我能积极融入大学生活，结识不同的朋友，在学习和生活的困境中能够坚强并坚持下来，到现在能够完全独立生活。感谢自己不负大学。