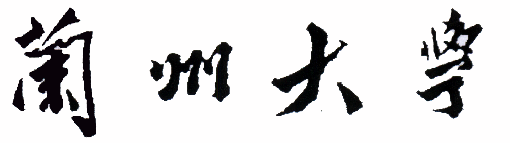
****

**毕业论文**

（本 科 生）

**论文题目（中文）** **光伏板功率监测系统设计与实现**

**论文题目（外文） The Design And Implementation of**

**Photovoltaic Board Power**

**Monitoring System**

**学 生 姓 名 蔡睿**

**导师姓名、职称 张杲峰讲师**

**学生所属学院 信息科学与工程学院**

**专 业 计算机科学与技术**

**年 级 2017级**

兰州大学教务处

**诚信责任书**

本人郑重声明：本人所呈交的毕业论文（设计），是在导师的指导下独立进行研究所取得的成果。毕业论文（设计）中凡引用他人已经发表或未发表的成果、数据、观点等，均已明确注明出处。除文中已经注明引用的内容外，不包含任何其他个人或集体已经发表或在网上发表的论文。

本声明的法律责任由本人承担。

论文作者签名： 日 期：

**关于毕业论文（设计）使用授权的声明**

本人在导师指导下所完成的论文及相关的职务作品，知识产权归属兰州大学。本人完全了解兰州大学有关保存、使用毕业论文的规定，同意学校保存或向国家有关部门或机构送交论文的纸质版和电子版，允许论文被查阅和借阅；本人授权兰州大学可以将本毕业论文的全部或部分内容编入有关数据库进行检索，可以采用任何复制手段保存和汇编本毕业论文。本人离校后发表、使用毕业论文或与该论文直接相关的学术论文或成果时，第一署名单位仍然为兰州大学。

本毕业论文研究内容：

□可以公开

□不宜公开，已在学位办公室办理保密申请，解密后适用本授权书。

（请在以上选项内选择其中一项打“√”）

论文作者签名： 导师签名：

日 期： 日 期：

**光伏板功率监测系统设计与实现**

# 摘 要

近年来，为了充分利用太阳能这种分布广泛、安全清洁的能源，光伏发电技术在我国得到了快速发展。我国甘肃酒泉地区太阳辐射强，光照资源好，潜在的太阳能发电装机容量可达到1.36亿万千瓦，年发电量可达2000多亿千瓦时。由于甘肃地区所处的黄土高原常年受风沙侵蚀、沙尘暴频发，光伏板表面在野外环境中容易积累灰尘，使得透光率下降，最终严重影响光伏发电效率。

为了探究光伏板表面灰尘的积累对光伏发电功率的影响，有必要监测光伏板表面的灰尘积累情况，并对光伏板的发电功率及相应的气象数据进行实时监测与记录。光伏发电与灰尘积累监测系统主要包括光伏发电台站及其主控模块以及服务端软件两部分，本文使用增量模型完成了对服务端软件的需求分析、概要设计以及部分功能模块的编码、部署与测试工作。本文通过对服务端软件进行需求分析，基于Node.JS设计一个借鉴微服务设计思想光伏发电监测系统服务端软件，为光伏发电台站和研究人员提供包括图片文件和发电数的据传输与存储、数据可视化、数据查询与下载等功能的可扩展的应用框架。

**关键词：**光伏发电；功率监测；气象监测；增量模型；监测系统；微服务

**THE DESIGN AND IMPLEMENTATION OF PHOTOVOLTAIC BOARD POWER MONITORING SYSTEM**

# Abstract

In recent years, photovoltaic power generation, as a widely distributed, safe and clean energy, has been developing rapidly in China. The Jiuquan area of Gansu Province in China has strong solar radiation resources. The potential installing capacity of solar power generation stations can reach 136 million kilowatts, and the annual power generation can reach more than 200 billion kilowatt hours. However, dust accumulation on photovoltaic panels surfaces is inevitable, which will cause a signiﬁcant decrease on the light transmittance and eventually affects the efﬁciency of photovoltaic power generation.

In order to explore the impact of the accumulation of dust on the surface of photovoltaic panels to power generation, it is necessary to monitor the accumulation of dust on the surface of photovoltaic panels, and to monitor and record the power generation of photovoltaic panels along with related meteorology data in real time. Photovoltaic power generation and dust accumulation monitoring system mainly includes two parts: photovoltaic power generation station and its main control module and server software. This article uses an incremental model to complete the demand analysis of server software, outline design, coding , deployment and testing of some functional modules. Based on the demand analysis of server software, this paper designs a photovoltaic power generation monitoring system server software based on Node.JS that draws on the micro-service design idea, providing photovoltaic power stations and researchers with a scalable application framework including transmission and storage of image ﬁles and data, data visualization, data query and download.

**Key words:** Photovoltaic Power Generation; Power Monitoring; Meteorology Monitoring; Incremental Model; Monitoring System; Micro-service

# 目 录

[摘 要 I](#_Toc73184559)

[Abstract I](#_Toc73184560)

[第一章 绪 论 1](#_Toc73184562)

[1.1 课题研究背景与意义 1](#_Toc73184563)

[1.2 国内外研究现状 2](#_Toc73184564)

[1.3 本文研究内容 3](#_Toc73184565)

[1.4 本文结构 3](#_Toc73184566)

[第二章 光伏板功率监测系统概述 4](#_Toc73184567)

[第三章 光伏板功率监测系统服务端 6](#_Toc73184568)

[3.1 需求分析 6](#_Toc73184569)

[3.1.1 功能性需求分析 6](#_Toc73184570)

[3.1.2 非功能性需求分析 8](#_Toc73184571)

[3.2 服务端软件设计 9](#_Toc73184572)

[3.2.1 软件框架结构 9](#_Toc73184573)

[3.2.2 服务端概要设计 10](#_Toc73184574)

[3.2.3 技术选型 12](#_Toc73184575)

[3.2.4 原型系统详细设计与实现 18](#_Toc73184576)

[3.3 原型系统部署与测试 27](#_Toc73184577)

[3.3.1 GET方式发送数据测试 29](#_Toc73184578)

[3.3.2 POST方式发送数据测试 29](#_Toc73184579)

[3.3.3 测试结果 30](#_Toc73184580)

[第四章 结论与展望 32](#_Toc73184581)

[参考文献 33](#_Toc73184582)

[附 录 34](#_Toc73184583)

[致 谢 40](#_Toc73184584)

# 第一章 绪 论

## 1.1 课题研究背景与意义

当前，人类的能源主要依赖于化学燃料，随着社会发展和环境形势的压力增大，未来人类必将转向开发使用可再生清洁能源[[1](#_bookmark40)]。太阳能一直被作为重点研究的新能源之一，具有取之不尽、分布广泛和安全清洁等优势。根据国家能源局关于光伏发电建设运行信息简况的统计数据，在2020年上半年[[2](#_bookmark41)]，我国太阳能发电容量达到21582万*kW*，其中集中式太阳能发电站14875万*kW*，分布式光伏发电站6707万*kW*。上半年光伏发电量1278亿kWh，同比增长20%。受光伏发电站所在场所的地理与气候条件影响，光伏板表面会出现灰尘积累，直接影响光伏发电的效率，需要对光伏板表面的积灰情况进行监测并通过监测数据预测光伏板发电效率。

光伏发电是太阳能发电的一种方式，是基于光伏半导体材料的光生伏打效应而将来自太阳的电磁波辐射能转化为直流电的现象创造的设施。太阳能电池板是光伏发电设备的重要部件，主要由单晶硅、多晶硅、非晶硅及碲化镉等半导体材料制作而成。光伏发电产业在本世纪世界各国对清洁能源的探索而不断加大投入建设的条件下得到了迅速的发展[[3](#_bookmark42)]。光伏发电系统主要由太阳能电池板、光伏发电控制器以及逆变器组成。在规模上可以分为独立式光伏发电系统与并网光伏发电系统。

独立光伏发电系统的基本结构如图[1.1](#_bookmark4)所示。它的输出通常被直接传输给用电器或者电池等储能设备，常用于解决电力基础设施建设不发达的地区用电难的问题。

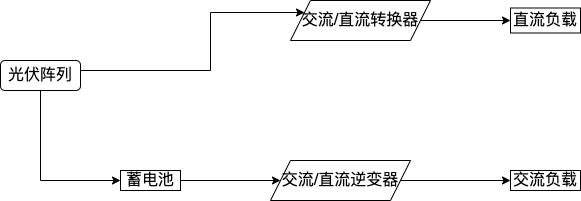


图1.1 独立光伏发电系统基本结构

并网光伏发电系统则需要经过逆变器等整流转换模块处理后产生符合国家电网要求、可直接连入电网的交流电，其基本结构如图[1.2](#_bookmark5)所示。

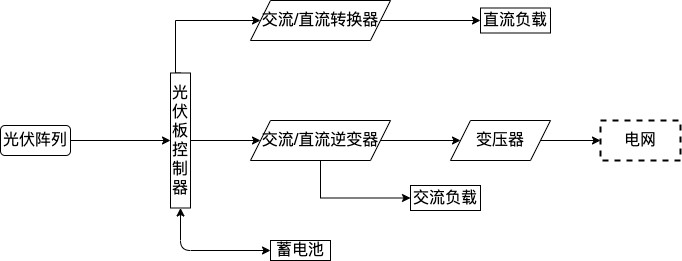


图1.2 并网光伏发电系统基本结构

光伏发电系统的大规模发展亟需解决的一大难题与发电效率密切相关：受到光伏电站所在区域的地理环境和气候条件影响，随着光伏板使用时间的推移，光伏板的玻璃面板会逐渐累计一层灰尘。导致太阳光的透过率下降从而降低发电效率。此外，光伏发电产出极度依赖于发电站所在地的太阳能资源与气候条件[[4](#_bookmark43)]，且初期建设需要大量资源的投入，成本下降还需要一定时间[[5](#_bookmark44)]。

根据现有研究，光伏板的发电效率会随着表面灰尘的积累量的增加逐渐下降，下降幅度浮动在2%−25%之间，一场沙尘暴后光伏板的发电效率可降低大约20%[[6](#_bookmark45)]。因此，通过对光伏板表面积灰情况以及发电效率进行实时监测，可以确定合适的清洗、维护光伏板的时机，最终能够达到显著延长光伏发电部件的使用寿命并提高光伏板生命周期内的电能产出的目的。

## 1.2 国内外研究现状

光伏板的发电效率受到包括环境温度、环境湿度、风速、风向、光伏板表面积灰度、太阳高度与方位以及太阳辐射状况在内的多种因素影响。

近年来国内外有较多专注于浮尘影响因素的实验研究。为了探明光伏板表面长期积灰的理化特性，SyedA.M等人[[7](#_bookmark46)]使用了包括电子显微镜、原子力显微镜和能量色散光谱研究了尘埃颗粒的特征以及尘埃在潮湿环境下形成的泥浆与干泥的化学力学，分析了潮湿环境下积灰和泥浆对光伏组件的影响并探讨了浮尘的清理方案。XueqingLiu等人[[8](#_bookmark47)]则是把侧重点放在工业污染产生的浮尘进行分析，通过对光伏板积灰进行分级对照实验，研究了工业产生的空气污染形成的粉尘微粒在不同气候环境对光伏发电效率的影响。此外，为了探究风速与粉尘积累的关系，赵明智等人[[9](#_bookmark48)]设计了风洞试验研究了沙尘对光伏组件输出特性的影响，实验表明在低风速条件下积沙对倾角组件输出功率影响不显著，而风速提升后积沙的对影响约放大5%。余操等人[[10](#_bookmark49)]专注于积灰清理策略的探究，对地面光伏电站的2个采取不同清洁策略方阵的运行数据和辐照数据分成上午数据和下午数据，分别进行处理在200W/m2以上的辐射强度下，上午的清洁增益约3.3%±2.1%，下午为3.6%±2.0%。

为了监测新能源发电系统的运行状态并进行维护，同时采集数据进行量化分析，国内外在上世纪90年代开始重视监测系统在新能源领域的应用。研究人员在开展了多项研究试验后逐渐形成了比较完善的光伏发电监测系统技术方案。莫康信等人[[11](#_bookmark50)]开发了一套基于ATmega单片机方案,由数据采集控制电路、LCD液晶显示与操作面板模块以及人机交互界面组成的光伏电站的环境参数监测系统。周念成等人[[12](#_bookmark51)]公开了一种光伏并网发电在线监测与状态评估系统的设计与实现方案，并提出了间谐波测量算法和谐波源定位与责任划分方法，实现对光伏并网发电站的电能质量、运行状态进行在线实时的全面监测。

## 1.3 本文研究内容

本课题为了研究远程监测光伏板浮尘对光伏发电的影响，为此在实验室搭建了一套具有采集发电数据、气象环境数据以及灰尘积累图像数据功能光伏发电实验平台，包括在光伏发电台站实现数据采集与发送的主控模块以及服务端软件。服务端软件实现了数据接收与存储、数据可视化以及数据查询功能。

本文的工作重心集中在使用增量模型对服务端软件的需求分析、设计与部分功能模块的实现以及部署测试上。

## 1.4 本文结构

本文在第二章对简要介绍了实验室光伏发电台站实验平台需要采集的数据；第三章则首先通过用例分析的方式对服务端软件进行需求分析，随后对服务端软件的整体架构进行了概要设计，通过需求和概要设计的特点确定了开发的技术选型并针对部分模块做出了详细设计与实现，最后展示了部分模块的部署测试结果；第四章总结了本文所做的工作，并对本文内容参与未来的研究活动进行了展望。

# 第二章 光伏板功率监测系统概述

本项目的最终目的是实现一个具有对光伏发电功率以及光伏板表面灰尘积累情况进行实时监测的系统，主要包括光伏发电台站软硬件和服务端软件两大部分，项目的整体结构层次如图所示。

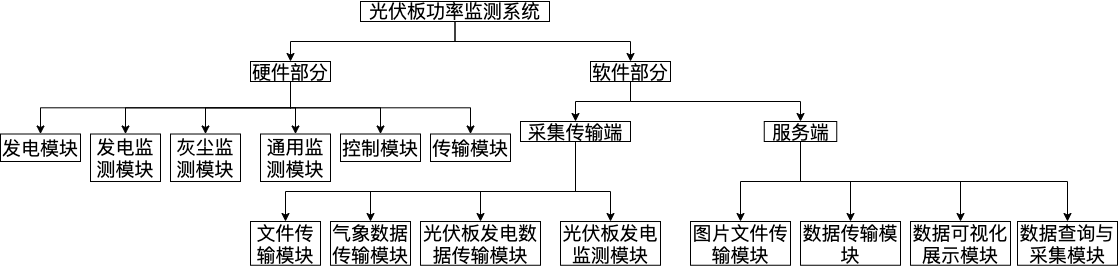


图2.1 光伏板功率监测系统层次结构

本项目的实施有三个主要目的：首先是为后期基于光伏板表面积灰度图像进行光伏板发电效率预测模型的训练提供一个数据采集平台，要求能够从光伏发电台站实时获取光伏板表面的照片、光伏板的发电状态以及光伏发电台站所在位置的气象条件数据并进行存储；其次，为了将采集到的数据用于后续的研究，有必要为研究人员提供能够方便地获取数据集的接口；最后，本项目中的监测系统需要大范围地安装在多个光伏发电厂的每个发电台站进行监控，因此有必要为国家电网非技术背景的工作人员以及用户提供易于使用的时空数据可视化展示界面。

监测系统分为上位机软件和服务端软件。上位机软件负责在光伏发电台站采集光伏发电数据和光伏板表面图像并向服务器发送。为了实现数据的采集、整合以及传输，需要安装在每个光伏发电台站的开发板上作为光伏发电站的主控模块，连接光伏板控制器以及气象站收集发电效率数据以及气象环境数据。主控模块在完成了一个数据采集周期后为采集到的数据与文件添加时间戳，随后与服务器建立连接进行数据传输，服务器则负责将主控模块传输的数据存入文件系统与数据库。

光伏板控制器与气象站收集的数据如表[2.1](#_bookmark10)中清单所示。

表2.1 光伏发电台站数据清单

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 数据来源 | 数据名称 | 计量单位 | 值类型 | 备注 |
| 光伏板控制器 | 数据采集器中断信号 | / | 布尔值 | 每 1min 采样一次 |
| 光伏板控制器 | 数据采集器电源告警 | / | 布尔值 | 每 1min 采样一次 |
| 光伏板控制器 | 光伏板输出电压 | *V* | 浮点数 (保留两位小数) | 每 1min 采样一次 |
| 光伏板控制器 | 发电量 | *kW·h* | 浮点数 (保留两位小数) | 每 1min 采样一次 |
| 气象站 | 1min 平均风速 | *m/s* | 浮点数 (保留两位小数) | 每 1min 采样一次 |
| 气象站 | 1min 平均风速 | *rad* | 浮点数 (保留两位小数) | 每 1min 采样一次 |
| 气象站 | 风速标准偏差 | / | 浮点数 (保留两位小数) | 3s 计算一次气 |
| 气象站 | 风速标准偏差 | / | 浮点数 (保留两位小数) | 3s 计算一次气 |
| 气象站 | 最大风速 | *m/s* | 浮点数 (保留两位小数) | 3s 计算一次气 |
| 气象站 | 最小风速 | *m/s* | 浮点数 (保留两位小数) | 3s 计算一次气 |
| 气象站 | 最大风向 | *rad* | 浮点数 (保留两位小数) | 3s 计算一次气 |
| 气象站 | 最小风向 | *rad* | 浮点数 (保留两位小数) | 3s 计算一次气 |
| 气象站 | 温度 | 摄氏度 | 浮点数 (保留两位小数) | 每 1min 采样一次 |
| 气象站 | 相对湿度 | *RH*% | 浮点数 (保留两位小数) | 每 1min 采样一次 |
| 气象站 | 气压 | *Pa* | 浮点数 (保留两位小数) | 每 1min 采样一次 |
| 气象站 | 全辐射 | *W/m*2 | 浮点数 (保留两位小数) | 每 1min 采样一次 |
| 气象站 | 反射辐射 | *W/m*2 | 浮点数 (保留两位小数) | 每 1min 采样一次 |
| 气象站 | 组件温度 | 摄氏度 | 浮点数 (保留两位小数) | 每 1min 采样一次 |
| 摄像头 | 光伏板表面照片 | / | 二进制文件数据 | 每 1min 采样一次 |

显然从表[2.1](#_bookmark10)中不难发现，不同类别的采样频率并不完全一致，为了保证单条数据记录的完整性，规定统一每分钟采样一次。

服务端软件负责接收光伏发电台站主控模块传来的图像文件与数据存入文件系统与数据库管理系统，数据的采集与可视化展示功能也由服务端软件提供。

从本项目的分工出发，本文将着重描述服务端软件的需求分析、各功能模块以及服务端整体软件架构的概要设计、技术选型以及部分功能模块的编码实现，并展示部署测试结果。

# 第三章 光伏板功率监测系统服务端

为了对服务端软件作出合理的设计与实现，结合本项目所要求设计的光伏发电监测系统的特点，本文采用增量模型进行设计与开发。通过分析，将服务端需要执行的任务分解成多个功能模块，各自进行设计、编码与测试。

## 3.1 需求分析

本文使用 Draw.io 与Visual Paradigm 建模工具进行需求分析、概要设计与详细设计。首先进行用例分析。

### 3.1.1 功能性需求分析

在本系统中，光伏发电与浮尘监测系统的服务端应包含以下功能:

1.光伏板图片文件传输功能

2.光伏发电台站数据传输与存储功能

3.数据可视化展示功能

4.数据查询功能

对于光伏发电台站的主控模块和研究人员而言，上述模块是服务端软件提供的交互接口，封装了对服务器文件系统和数据库的交互功能。研究人员和主控模块对服务端软件的交互用例如图[3.1](#_bookmark14)所示。

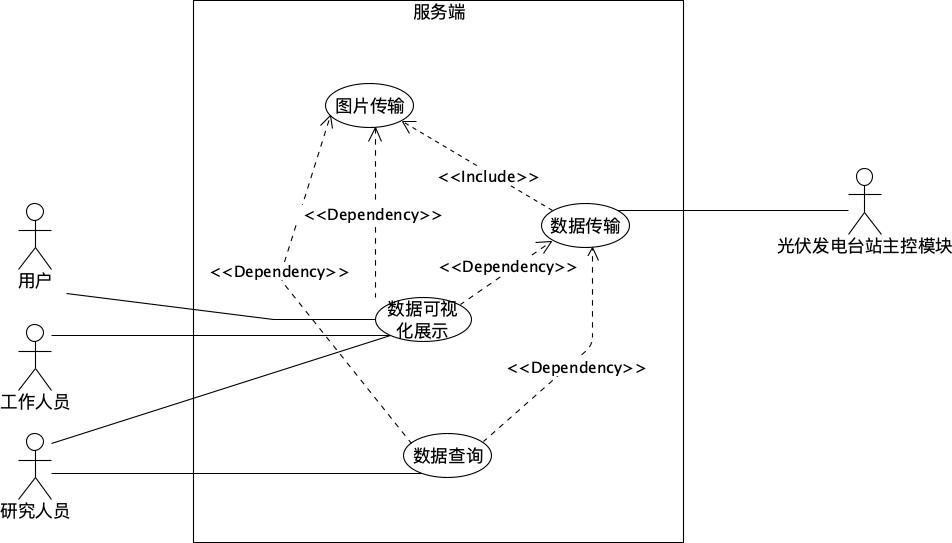


图3.1 服务端软件用例图

用例图中描绘的用例是从服务端软件必须为使用方提供的各个功能中抽象出来的，这些用例能够比较好地反映出使用方的需求。对这些用例进行精细化的描述后可以得到如下用例的详细描述：

表3.1 图片传输用例

|  |  |
| --- | --- |
| 用例名 | 图片传输 |
| 用例目的 | 服务器接收并存储光伏发电台站发送的光伏板表面照片 |
| 参与者 | 光伏发电台站, 服务器及其文件系统 |
| 前提事件 | 需要光伏发电台站的主控模块对服务器有上传文件的权限 |
| 事件流 | 1.光伏发电台站的主控模块与服务器建立连接 |
| 2.主控模块向服务器传输图片文件 |
| 3.服务器将图片存储到特定的目录下 |

表3.2 数据传输与存储用例

|  |  |
| --- | --- |
| 用例名 | 数据传输与存储 |
| 用例目的 | 服务器接收光伏发电台站发送的各项数据 |
| 参与者 | 光伏发电台站, 服务器及其数据库管理系统 |
| 前提事件 | 需要服务器已经完成此次数据发送前光伏发电台站发送的图片数据的接收和存储 |
| 事件流 | 1.光伏发电台站的主控模块与服务器建立连接 |
| 2.主控模块向服务器传输打包后的各项数据 |
| 3.服务器检测文件系统中是否已存储对应的光伏板图片 |
| 4.服务器对数据的完整性与合法性进行检查 |
| 5.服务器将数据以合适的范式作为记录存储到数据库中 |
| 备注流程 | 1.图片不存在时放弃存储数据，要求主控模块进行图片和 数据的重发 |
| 2.数据内容不合法放弃存储数据，返回数据传输失败的日志信息 |

表3.3 数据可视化展示用例

|  |  |
| --- | --- |
| 用例名 | 数据可视化展示 |
| 用例目的 | 对各地的光伏电站发电效率以及积灰情况作直观的展示 |
| 参与者 | 研究人员，工作人员，用户，服务器数据库以及文件系统 |
| 前提事件 | 服务器已经接收了待展示的光伏发电台站数据以及光伏 板表面图片 |
| 事件流 | 1.用户向服务器请求光伏发电台站的数据 |
| 2.服务器将请求的发电台站的地理坐标通过地理信息组件渲染到地图上进行位置展示 |
| 3.服务器将光伏发电台站的发电效率以及积灰度等历史数 据通过图表等形式进行编排展示 |
| 4.服务器向研究人员返回图形化展示页面 |

表3.4 数据查询用例

|  |  |
| --- | --- |
| 用例名 | 数据查询 |
| 用例目的 | 针对用户请求的光伏发电台站返回对应的历史发电数据以及图片文件 |
| 参与者 | 研究人员，服务器数据库以及文件系统 |
| 前提事件 | 服务器已经接受了研究人员请求的光伏发电台站数据以及光伏板表面图片 |
| 事件流 | 1.研究人员向服务器请求光伏发电台站的数据 |
|  | 2．服务器将请求的发电台站的历史发电效率数据以及 每条记录所对应的光伏板表面图片发送给研究人员 |

### 3.1.2 非功能性需求分析

为了提高使用服务端在长期大量接收存储光伏发电以及积灰数据的可靠性，服务端软件还需要考虑以下非功能性需求：

1.性能需求：

（1）吞吐量：由于光伏发电场广泛布置于不同的地区，而每个光伏发电场又有大量的 发电台站，每个台站的主控模块每分钟都会向服务端发送包括图片文件在内的 数据。根据[3.5](#_bookmark21)，假设摄像头拍摄的图片文件平均大小为 5MB，河西走廊地区的光伏电场数量按照 184 个计算，平均每个光伏电场有 10 个光伏发电台站，那么每分钟所产生的数据量为(5 × 220 + 4 × 16 + 2) × 184 × 10 = 9647020640B ≈ 9200MB，则吞吐量最大可达 9647020640 × 8 ÷ 60 ≈ 1286269418.6bps ≈ 10130Mbps。服务端软件需要对高并发的大吞吐量数据进行有效处理。

（2）响应时间：光伏发电与积灰情况的每组记录都包含图片文件和数值型数据，而且需要在完成图片的存储后再接收数据，所以在一次数据接收流程中服务端必须确保在接收图片后及时向主控模块返回响应并请求数据。

2.可拓展性与可维护性：

光伏发电场作为重要的基础设施建设，发电效率数据可用于后续其他专题实验， 应确保监测系统能对需求的不断变动作出快速响应，当需要对监测系统增添插 件时不需要对现有系统结构和代码做过多修改。

3.可靠性：

系统出错时不影响用户的行为与操作数据，例如光伏发电场通常设置于移动网 络基础设施建设完备度较低的野外，需要依赖 GPRS 远程通信模块进行网络访问，当发电台站主控模块连接中断一段时间后恢复链接，服务端需要请求其对 断网期缓存间的数据进行重新发送。

## 3.2 服务端软件设计

### 3.2.1 软件框架结构

基于光伏发电及积灰监测系统的用例需求，本系统采用如图[3.2](#_bookmark18)所示的功能架构：

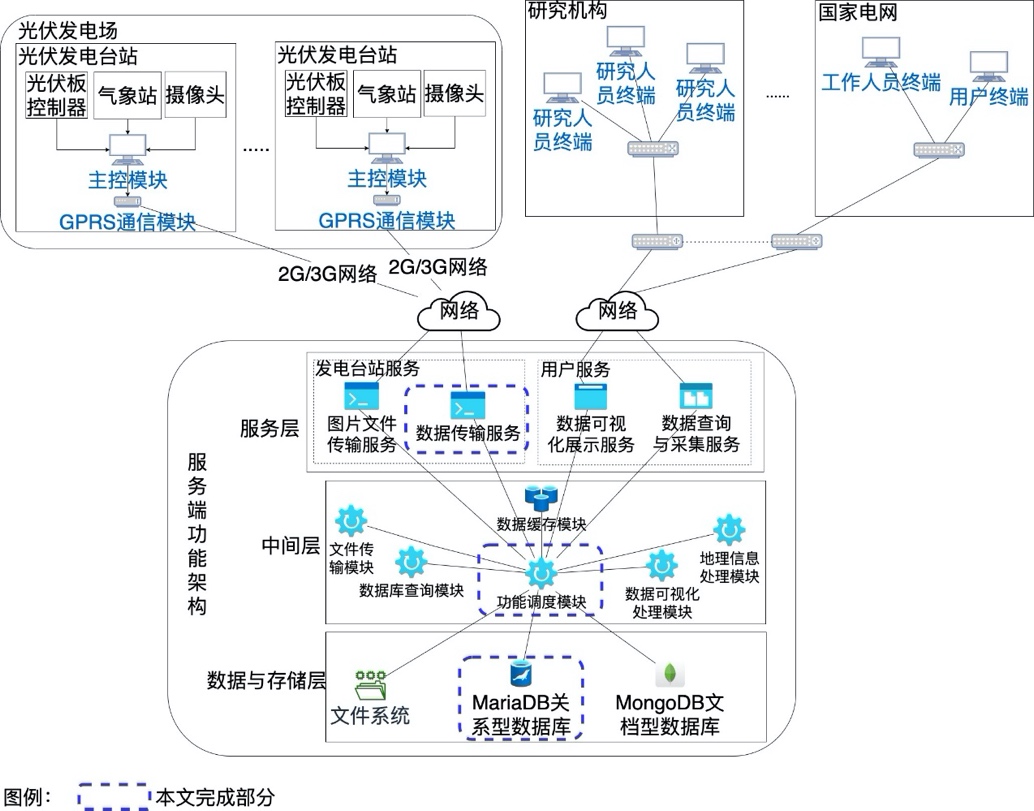


图3.2 服务端功能架构图

本项目将服务端的软件功能架构从逻辑上进行了分层。服务层中包含各个服务功能模块的具体实现，包含了处理来自使用方不同服务请求的业务逻辑，其中发电台站服务包括发电台站主控模块需要使用的图片传输服务以及数据传输服务，用户服务则包含研究人员需要使用的数据查询服务以及国家电网工作人员以及国家电网客户使用的数据可视化展示服务；中间层为服务层提供了封装了多种功能的模块化接口，并且对各模块I/O和缓存进行功能调度；数据与存储层为中间层的模块提供文件系统和数据库管理系统的读写接口。

为了实现降低服务端各功能模块的耦合，使整体软件结构更易于维护，服务端在功能 结构设计上借鉴了微服务设计思想。此结构把与服务端进行通信的客体进行分类并提供不同类型的服务，具有如下优势：

1.对于研究人员而言，提供给他们使用的借口仅有数据的可视化展示和数据查询服务，开发人员能够重复利用网页端和移动端的业务逻辑代码。

2.确保各功能模块的边界明晰，功能归属秩序合理。防止单个功能模块为了给其他模块 提供借口而引入过多逻辑而变得臃肿。

3.数据统一经过功能调度模块进行传输和共享，可以避免接口调用关系混乱的问题。

4.对不同的数据选择性地进行冗余备份存储，放置于其他数据库，使单一数据库不被过 多服务依赖，降低后续重构与优化难度。

### 3.2.2 服务端概要设计

为了更好地对使用方与服务端软件之间的交互逻辑进行建模，本节使用了顺序图对提供服务过程中服务层应用模块与下层接口对象之间的消息传递进行分析，有助于理清各对象的生命周期中的关键行为及其先后依赖关系，并为后续各服务模块的详细设计提供参考依据。

1.发电台站服务

光伏发电台站的主控模块以每分钟一次的频率将光伏板表面的照片以及采集到的各项数据通过图[3.2](#_bookmark18)中服务端的发电台站服务进行发送，发电台站服务包含图片文件传输服务和数据传输服务两部分，这两个服务在光伏发电台站主控模块的每次数据发送中都需要先后使用。服务顺序如图[3.3](#_bookmark20)所示。

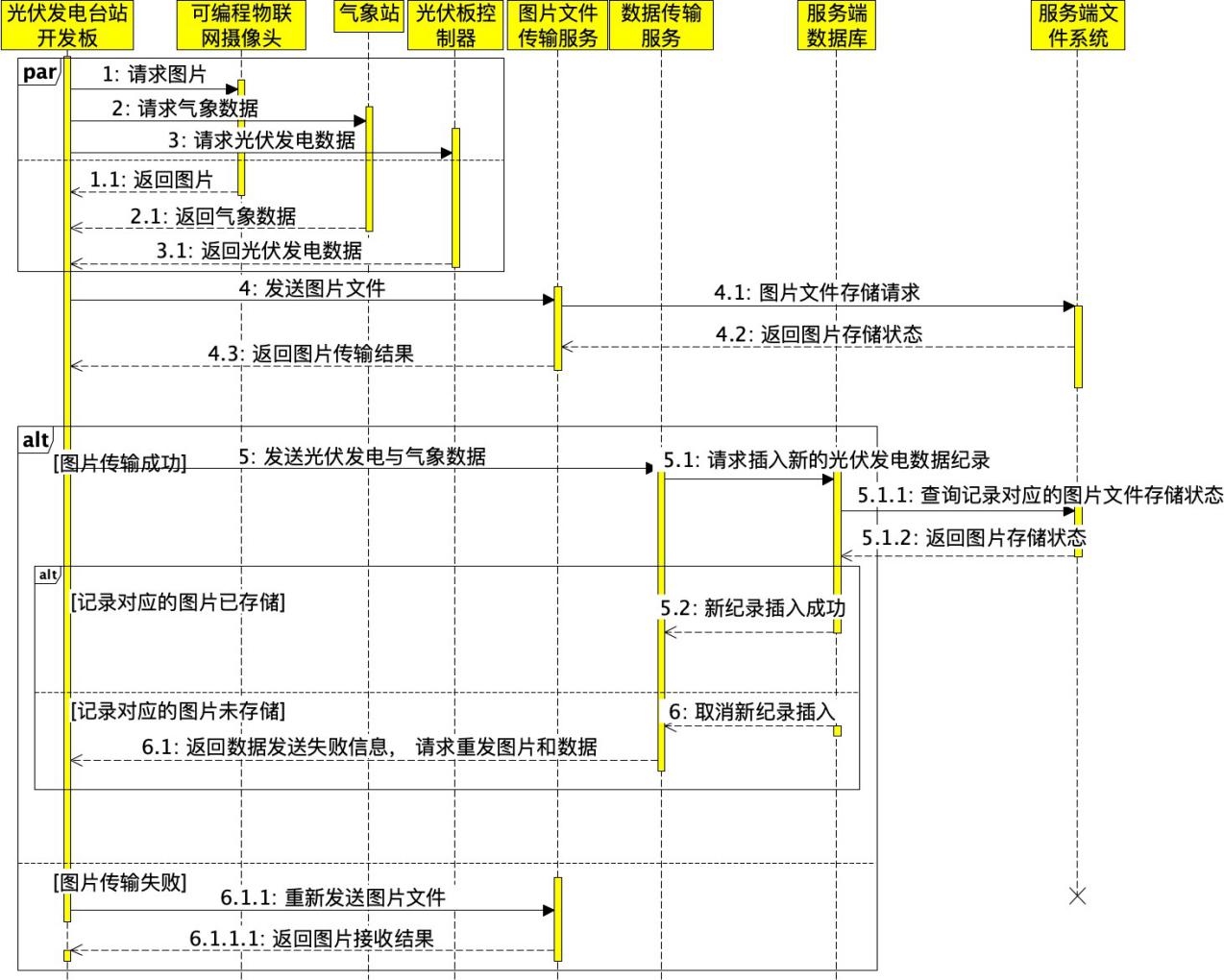


图3.3 光伏发电台站服务顺序图

对作为参与者的光伏发电台站主控模块而言，先后使用服务端的发电台站服务集合中的图片文件传输服务以及光伏发电与气象数据传输服务这两个项目的过程应被视为一种同步原子操作，即光伏板需要先完成了图片文件的传输才能进行与图片文件对应的数据记录的发送。

在每次使用服务端软件提供的发电台站服务前，主控模块需要首先完成所在光伏发电台站的各项数据采集，这些数据分别来自光伏板控制器、可编程物联网摄像头以及气象站，如顺序图[3.3](#_bookmark20)所示，该过程可由如下几条消息表示：

表3.5 光伏发电站顺序图消息说明

|  |  |
| --- | --- |
| 消息序号 | 消息说明 |
| （1） | 主控模块向可编程物联网摄像头发送请求返回光伏板表面图像的命令 |
| （1.1） | 可编程物联网摄像头完成拍摄后将获得的图片返回给主控模块 |
| （2） | 主控模块向气象站发送请求返回包括温度、湿度、风向、风速以及太阳辐射等项目在内的气象数据的命令 |
| （2.1） | 气象站完成采集后将数据返回给主控模块 |
| （3) | 主控模块向光伏板控制器发送请求光伏板输出电流、输出电压等工作状态数据的命令 |
| （3.1） | 光伏板控制器完成采集后将数据返回给主控模块 |

其中，对来自三个不同数据源的数据请求是异步操作，只有当完成所有的数据采集后主控模块才会进入下一步与服务器进行通信的流程。

2.数据查询服务

研究人员可通过数据采集与查询服务从服务端获取特定格式的数据集用于分析与研 究，数据查询服务的顺序如图[3.4](#_bookmark22)所示。

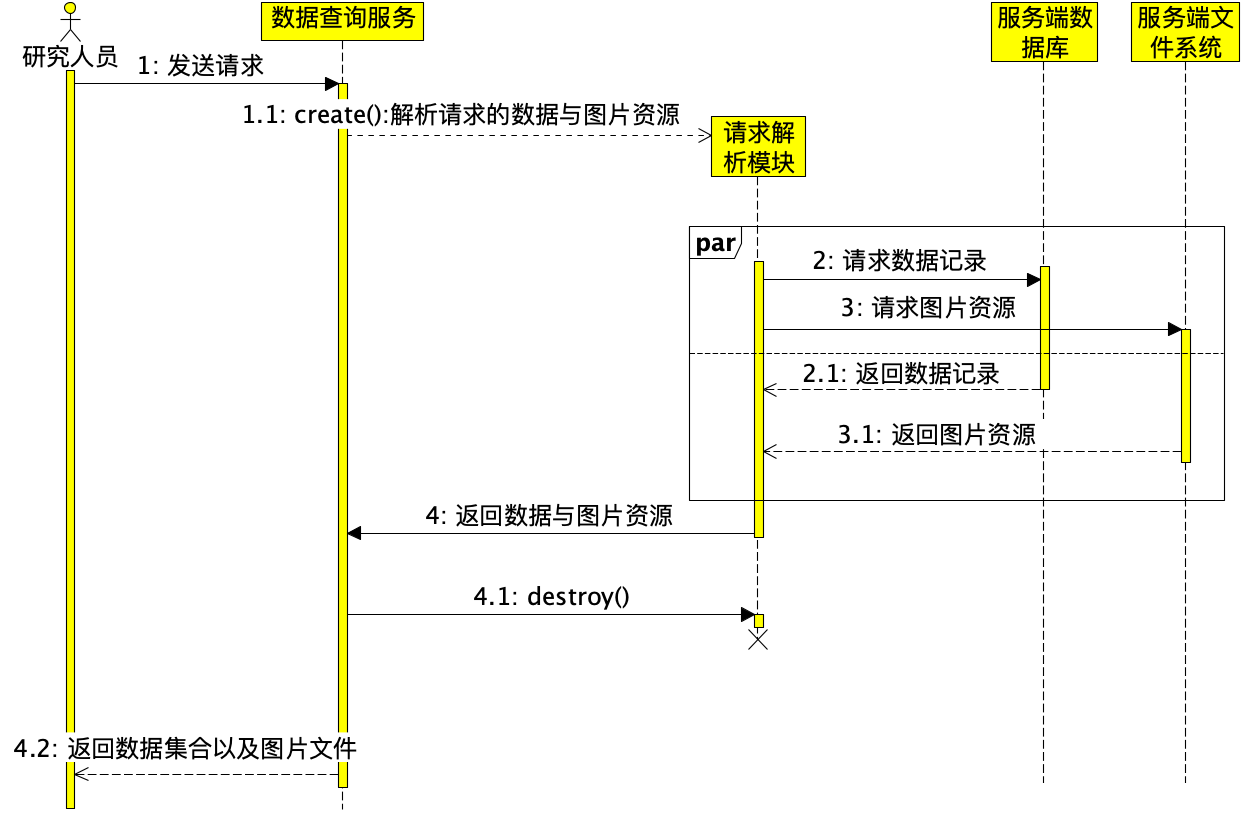


图3.4 数据查询服务顺序图

为了对多个研究人员执行者发送的请求，数据查询服务应针对每一个执行者创建解析模块协程，并由专门的线程池管理数据与图片文件的获取结果，最终在完成了所需资源的获取后及时销毁请求解析模块协程。

3.数据可视化展示服务

数据的可视化展示服务的工作顺序和数据查询服务大体相同，同样需要针对研 究人员的请求内容进行数据和图片资源的获取。不同点在于有必要增添一个可视化处理模 块用以生成图形化展示页面返回给研究人员。

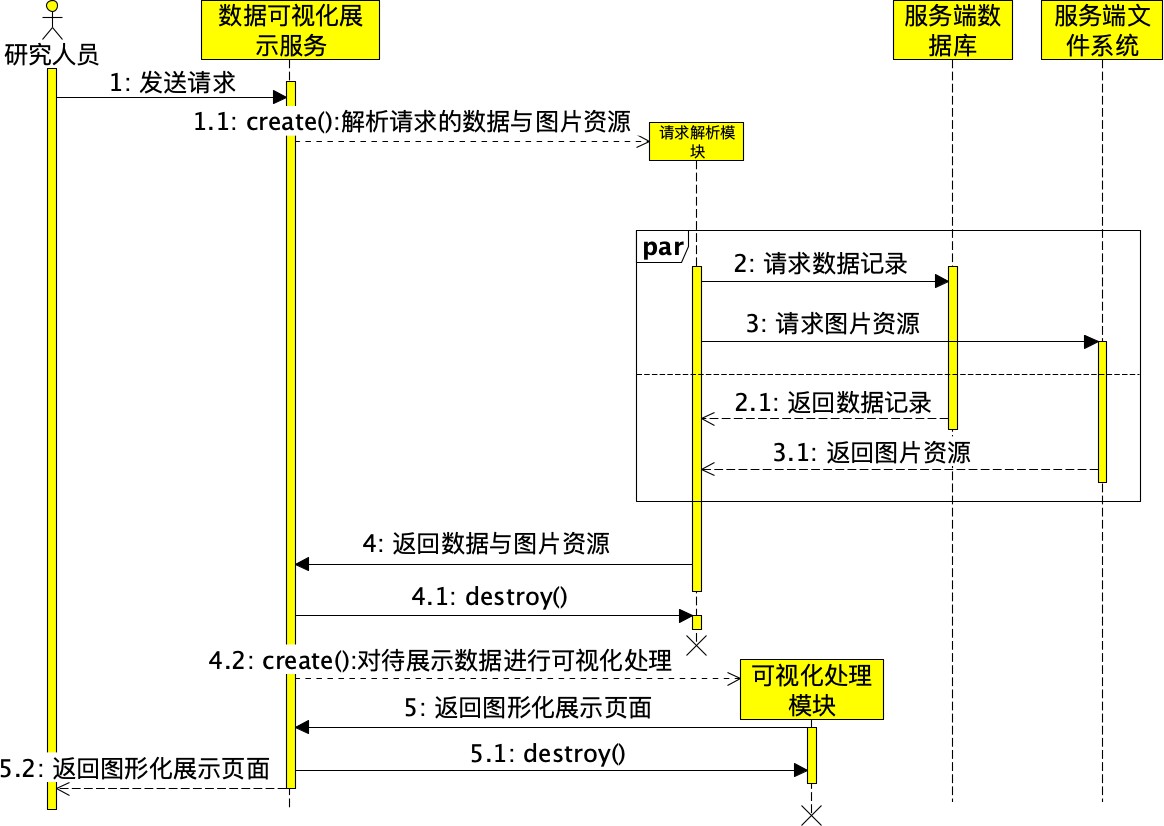


图3.5 数据可视化展示服务顺序图

### 3.2.3 技术选型

从服务层各模块的顺序模型中不难发现，为使用方提供的功能模块都包含涉及对文件 系统与数据库系统进行访问与读写的异步操作，而由于光伏发电场及其用户在政策的倾斜下有着规模不断扩大的趋势，服务端软件需要对可预见的数据吞吐量与用户访问请求量扩大留有拓展能力，并保证后续开发过程中的可维护性。本文针对本项目的特点，选取Node.JS 作为服务端软件的开发工具。

1.Node.JS 与 Express 框架

Node.JS 是基于Chrome V8 引擎开发的一个异步事件驱动的 JavaScript 运行时，它被设计用来构建可扩展的网络应用，与过去只能运行在浏览器环境中的传统 JavaScript 解释器不同，Node.JS 为开发者提供了对文件系统以及硬件资源的访问能力。[[13](#_bookmark52)] 为 Node.JS 提供文件系统以及硬件资源访问能力的组件是libuv，用于执行非阻塞型的 I/O 操作，封装了全部可支持的操作系统的调用接口。它提供了处理诸如DNS、网络、子进程、管道、信号量控制、轮询机制和数据流的机制，且同样也实现了一个管理非操作系统层级的异步任务的线程池。Node.JS 的架构如图[3.6](#_bookmark24)所示。

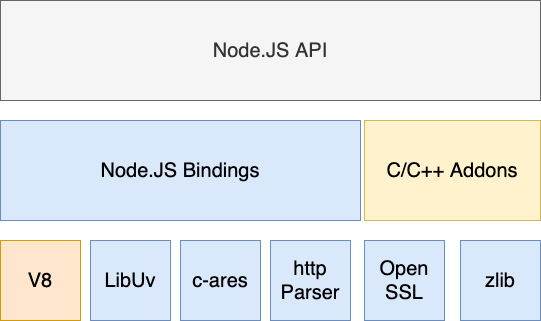


图3.6 Node.JS 的架构

由于Node.JS具有异步非阻塞I/O和事件驱动等特性，对数据密集型、I/O密集型以及高并发请求访问具有良好的处理能力。Node.JS单线程运行程序脚本，同时使用事件循环处理非阻塞I/O，将处理I/O事件的线程转移到操作系统内核。[[14](#_bookmark53)]

由于大多数现代操作系统内核都是多线程的，有能力在后台处理多个操作。当一个操作完成时，内核通知Node.JS把该操作对应的回调函数添加到轮询队列中等待执行。下面的图[3.7](#_bookmark25)展示了使用 node main.js 命令运行脚本后事件循环 (Event Loop) 操作顺序的简化概览。

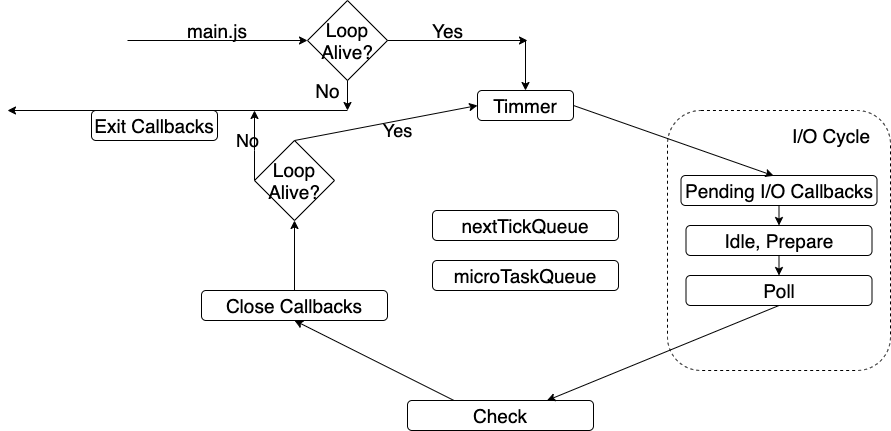


图3.7 Node.JS 事件循环(Event Loop) 机制图

图[3.7](#_bookmark25)中的每个圆角方框都代表了事件循环的一个阶段 (Phase)，每个阶段都有一个队列用于管理特定阶段新加入的回调函数等异步任务。下面是事件循环每个阶段的简要描述：

1.定时器（Timer）阶段：事件循环起始阶段，在脚本执行完成后立即进入，定时器阶段的回调函数队列中有定时器（setTimeout, setInterval）函数的回调函数，进入此阶段后到达回调函数规定的等待时间后执行。

2.处理中（Pending）的 I/O 回调阶段：此阶段执行在事件循环检查pending\_queue 中是否存在未处理的回调函数，发现未处理的函数后将依次执行，直到队列为空，或者达 到系统的最大限制。这些回调函数是被之前的操作推入的，例如当尝试往tcp 中进行写入，这个写入工作完成后回调函数就会被推入到队列中。错误处理的回调函数也在这里被执行。

3.空闲与准备（Idle, Prepare ）阶段：这两个阶段是node做一些重要的初始化内部操作的阶段。

4.轮询（Poll）阶段：这是事件循环最重要的一个阶段，这个阶段接受新传入的连接（例如新的 Socket 建立等）和数据读写（文件读取等）活动并对这些事件的回调函数推入合适的事件循环调度队列中，直到时间耗尽或者像其他阶段那样达到系统依赖上限。

5.检查（Check）阶段：轮询阶段结束之后，立即来到检查阶段。setImmediate 的回调函数参数会进入检查阶段的队列，该阶段同样会将队列中的回调函数依次取出执行，直到队列为空或者达到系统的最大限制。

6.关闭（Close）回调：处理关闭或者销毁类型的回调函数在关闭回调阶段执行，这一阶段在检查阶段结束后开始，这类回调函数通常被被绑定在消息类型为 close 的 Event 对象上（例如 socket.on("close",()=>...)）。事件循环执行完关闭阶段的队列中的回调函数后，如果事件循环中仍有未完成的事件将进入下一轮的循环，如果事件循环的任务都已完成则离开事件循环进入退出阶段。

7.nextTickQueue 与 microTaskQueue：这两个队列比较特殊，他们分别存储了由process.nextTick() 推入队列的回调函数以及 Promise 对象的 then() 方法推入的回调函数，这两个队列中的回调函数被设置为从当前阶段到下一个阶段之前尽可能快的运 行。不像其他阶段，这两个队列没有系统依赖的最大限制，node 不断运行这两个队列中的回调函数直到两个队列为空。

由此可见，Node.JS 中与 I/O 的交互都遵循异步编程的风格，I/O 操作的相关函数通常是以高度嵌套的回调函数（即一个函数作为另一个函数的参数）处理的，尽管这些 I/O 操作的函数也有同步执行的实现，在绝大多数情况下 Node.JS 的开发人员都不需要使用同步执行的函数。而在处理异步非阻塞 I/O 时则是由平台架构层libuv 灵活地调度操作系统提供的异步 API 或者线程池中的工作线程进行处理。[[15](#_bookmark54)]

Node.JS 本身自带支持构建 HTTP 的 API 模块，Express框架则是基于 Node.JS 开发的一个支持快速构建 HTTP 服务的框架[[16](#_bookmark55)]，它的 API 风格非常直观简单，具有详尽的文档，且由于支持开发人员自定义实现特定功能的路由中间件（Middle Ware）而具有灵活得可扩展性。

以下代码片段摘自本项目使用 Express 框架开发的 HTTP 服务器程序的入口点，完整文件可见附录：

const express = require('express'); //引入Express模块

const app = express(); //创建应用对象

...

//为应用对象注册一个监听访问根目录请求的回调函数

//设定默认的静态响应页面路径

app.use(express.static(path.join(\_\_dirname,'public')));

//该Web应用监听服务端的3000端口，开始监听后在终端输出日志信息

app.listen(config.port,() => console.log(`Server Running, Listening on ${config.port}...`));

运行后对服务器的3000 端口进行访问返回的响应如图[3.8](#_bookmark26)所示。



图3.8 服务端主页程序界面

其中调用 app.use(express.static(path.join( dirname,'public'))) 方法是使用了框架本身提供的中间件，中间件是 app 对象路由栈中的对象实例，它们的作用是将参数函数按照调用接口的性质推入事件循环中的处理中（Pending）的 I/O 回调阶段或者轮询（Poll）阶段，方法的参数则可以替换成针对不同路由字符串的自定义回调函数 [[17](#_bookmark56)]。由此可见，路由的设置方式确保了搭建针对不同访问 URL 处理逻辑的灵活性，为服务端软件的各个服务模块的可拓展性和可维护性提供了良好的支持。

2.数据库管理系统

本项目需要采集的数据清单如表[2.1](#_bookmark10)所示，然而为了对分散于各地的多个光伏电场以及每个光伏电场中的不同光伏发电台站进行管理，并且考虑到每个光伏发电台站每分钟都需要产生一次数据，对与数据存储方式的合理设计理应结合服务端的软件功能结构本身考虑。考虑到数据的用途和特点，可以将发电效率和气象的相关数据和地理信息数据分别放置于关系型数据库和文档型数据库。关系型数据库在本文实验中选用了MariaDB10.5.8，这是一个由开源社区采用GPL授权许可维护的一个关系型数据库，目的是在MySQL的版权被收购后能够保留一个源码开放的版本，因此MariaDB完全兼容MySQL的所有命令行与API。数据的实体关系如图[3.9](#_bookmark27)所示。

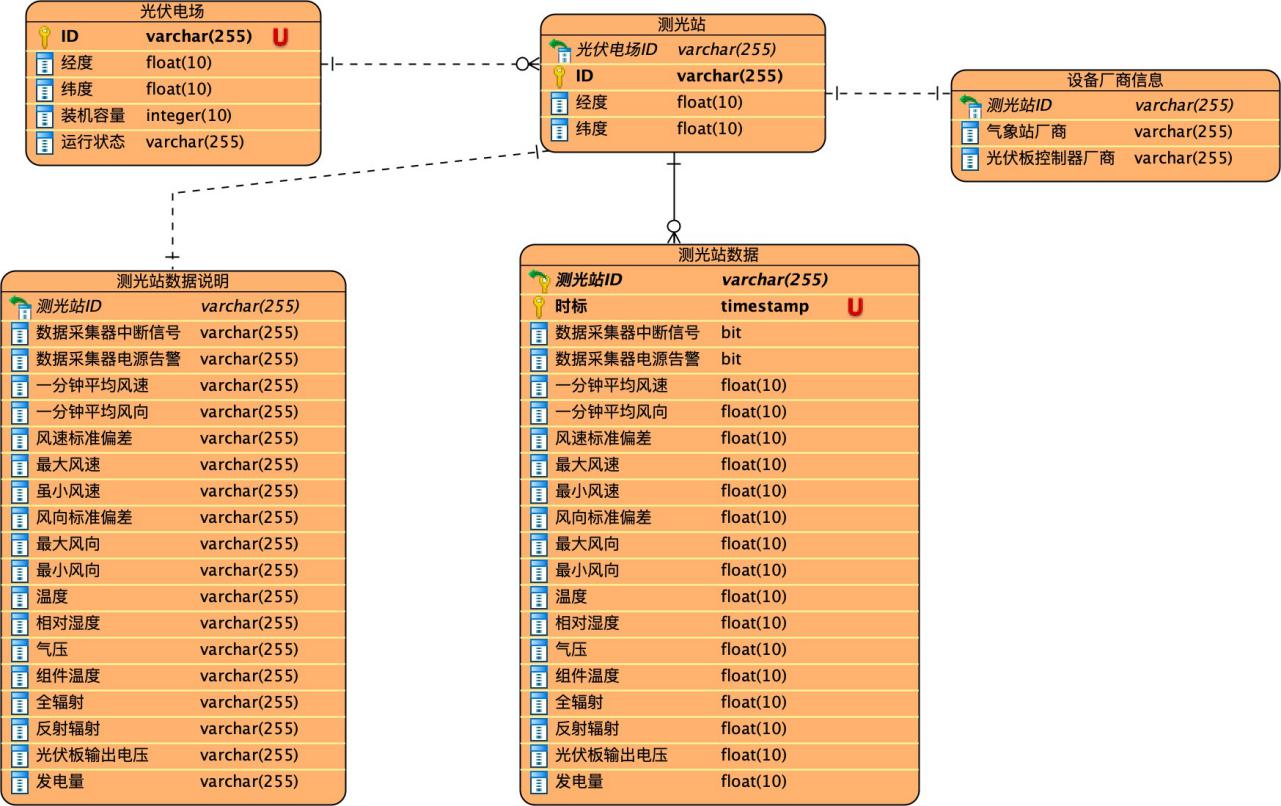


图3.9 关系型数据库实体关系图

另外，需要在数据库中存储高清摄像头拍摄的光伏板表面图片, 在关系型数据库中一般有两种做法：

1.将图片保存在文件系统中的路径存储到数据库。

2.将图片以二进制数据流的形式直接写入数据库字段中。

在形如 MySQL 的数据库系统中，尽管理论上能用 `blob(Binary Large Object)`容器将图片的二进制数据一并存入数据库中，但由于这样做会极大地增加数据库系统的负担，导致性能的下降与不稳定，所以更推荐采用的方式是将图片按照有规律的命名方式存储在一个固定的目录中再将其对应的路径作为字符串存入数据库.

由于提供给研究人员的可视化展示服务需要中间层的地理信息处理模块使用地理信息数据渲染展示页面，有必要将地理信息数据另外做一份存储。

GeoJSON 是一种在近年来被大量 Web 应用 API 所广泛使用的地理信息格式规范，支持将多种数据类型存储入 Javascript 对象标记模型（Javascript Object Notation）表示的树形结构体中[[18](#_bookmark57)]。下面是一个包含多个点坐标的地理信息对象格式：

{

"type": "FeatureCollection",

"features": [{

"type": "Feature",

"geometry": {

"type": "Point",

"coordinates": [102.0, 0.5]

},

"properties": {

"prop0": "value0"

}

}, {

"type": "Feature",

"geometry": {

"type": "LineString",

"coordinates": [

[102.0, 0.0],

[103.0, 1.0],

[104.0, 0.0],

[105.0, 1.0]

]

},

"properties": {

"prop0": "value0",

"prop1": 0.0

}

}, {

"type": "Feature",

"geometry": {

"type": "Polygon",

"coordinates": [

[

[100.0, 0.0],

[101.0, 0.0],

[101.0, 1.0],

[100.0, 1.0],

[100.0, 0.0]

]

]

},

"properties": {

"prop0": "value0",

"prop1": {

"this": "that"

}

}

}]

}

GeoJSON 将具有树形结构的半结构化数据存储为 JSON 格式的文本，这类数据通常使用文档行数据库进行管理。本项目选用 MongoDB 作为存储数据可视化展示服务所需的地理信息的管理系统，它能够高效地将地理信息数据按照 GeoJSON 标准存储，并大大简化与Web 应用之间的交互。

此外，可以考虑使用Redis 作为服务端中间层的数据缓存模块。Redis 作为一种内存键值存储数据库支持存储多种value 数据类型，包括链表、集合、字符串和哈希表等。由于将数据存储在内存中，在应对数据读取密集型的应用程序时能有效地提高读取效率。[[19](#_bookmark58)]

### 3.2.4 原型系统详细设计与实现

根据 Express 框架的特性，本项目将服务层各功能模块封装成可供 express.app 使用的中间件对象；功能调度模块的逻辑在服务端程序入口点源文件 app.js 中实现，负责对各功能模块中间件进行管理；中间层的其他功能模块则通过封装 Node.JS 标准库提供的非阻塞I/O 接口以及 npm package 中的接口实现。

1.数据查询服务设计

数据查询服务在接收到研究人员发送到服务对应路由下的请求报文后，首先解 析请求报文体中的查询字符串，根据请求报文中的查询目标尝试从数据库和文件系统中获取需要的数据与图片文件。当数据或图片文件中任一者获取失败时，回调函数Resource- QueryCallback() 对研究人员返回包含查询失败日志信息的 HTTP 响应报文；如果数据与图片文件获取成功，则将数据写入 csv 文件中并与图片文件一并打包成压缩文件通过 HTTP 响应报文发送给研究人员。服务所执行的操作主要在回调函数 ResourceQueryCallback() 中实现，伪代码表示如下：

/\*

函数名： ResourceQueryCallback

输入： http请求报文中的请求字符串

输出： 日志信息或包含表格文件和图片文件的压缩包

\*/

VAR isQuerySuccess : boolean //记录查询是否成功的布尔值

VAR isImageExist : boolean

PROC ResourceQueryCallback

(queryStr : stirng,

conn : 数据库连接对象(DatabaseConnection),

ImagePath : 图片文件存储路径(string),

outputDir : 输出文件路径(string),

fs : FileSystemAPI,

csv\_gererator : CSV文件生成器(CSV-Gernerator),

tar : 压缩文件生成器)

BEGIN

// 存储查询结果的数组

VAR queryResult : array[1..n] of records

conn.doQuery( queryStr, isQuerySuccess )

// 查询失败则直接返回错误信息

if isQuerySuccess == false then return QueryErrorMessage

// 查询成功时将记录写入csv文件

csv\_gererator.write(outputDir, 'output.csv', queryResult)

// 将记录对应的图片复制进输出目录，如果有文件读取失败则结束服务并返回错误信息

for i <- 1 to i == n do

isImageExist <- CheckImageExist(ImagePath, queryResult[i].imageName)

if not isImageExist then return FileErrorMessage

copy(ImagePath, queryResult[i].imageName, outputDir)

end

// 打包压缩文件并向客户端发送

tar.create(outputDir, 'output.tgz')

send('output.tgz')

delete(outputDir)

delete('output.tgz')

END

服务的活动图如图[3.10](#_bookmark29)所示。

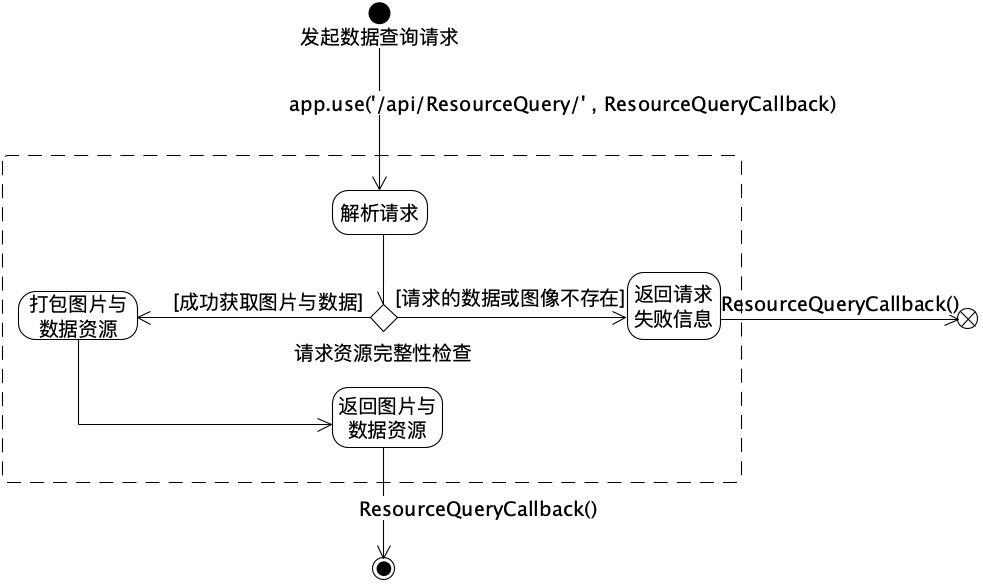


图3.10 数据查询服务活动图

实际项目代码中则分为两部分，首先是在 app.js 中将路由中间件绑定到对应的路由路径下：

const ResourceQueryCallback = require('./ResearchQeuryRouter');

...

// 数据查询路由中间件

app.use('/api/ResourceQuery/' , ResourceQueryCallback);

...

而回调函数则是封装在单独的路由中间件脚本文件 ResearchQeuryRouter.js 中：

/\* 引入模块常量 \*/

...

const pool = mysql.createConnection({

socketPath:config.DB\_socketPath,

user : config.DB\_user,

password : config.DB\_password,

database : config.DB\_database});

// 解析Post请求，返回请求中查询的数据与图像

router.post('/',(req,res)=>{

// 解析请求字符串

const queryStr = req.body.queryStr;

pool.query(queryStr,(err, result ,fields) => {

// 查询失败返回失败信息HTTP响应，结束本轮服务

if(err)

{

return res.json({msg : `查询失败，日志信息：\n${err}`});

}

// 查询成功则将数据写入csv文件并与图片打包返回

// 创建输出文件目录：

const outDir = path.join(\_\_dirname,'out',moment().format('yyyy-mm-dd:hh:mm:ss'))

fs.mkdir(outDir);

// 从图片文件目录下复制记录对应的图片文件到输出文件目录：

for(let i = 0; i < result.size; ++i)

{

fs.copyFile(path.join(config.imageDir, result[i].ImageFilename),

path.join(outDir,result[i].ImageFilename),data,

(err) => {

// 图片文件复制失败返回包含文件读写异常信息的HTTP响应，结束本轮服务

if(err)

{

return res.json({msg : `图片文件读取失败，日志信息：\n${err}`});

}

});

}

// 创建csv文件写入流：

const outFileStream = fs.createWriteStream(path.join(outDir, 'out.csv'));

csv\_generate({columns : result, length : result.length}).pipe(outFileStream);

// 压缩文件输出目录

const tarFileName = outDir + '.tgz';

tar.c(

{

gzip: <true|gzip options>,

file: 'my-tarball.tgz'

},

[outDir]

).then(\_ => {

// 在响应中发送打包后的文件

res.send(tarFileName);

}).catch( {err} => {

// 打包失败则发送带有错误信息HTTP响应，结束本轮服务

return res.send({msg : `文件打包失败，日志信息:${err}`});

} );

})

fs.rmdir(outDir,(err)=>{ if(err) console.log('删除临时目录失败') });

});

module.exports = router;

由于服务模块的主要功能实现都被封装在单独的脚本文件中被作为回调函数导出，为了方便描述，后续的服务详细设计将仅展示回调函数的伪代码。

2.数据可视化展示服务设计

数据可视化展示服务在接收到用户发送的请求报文后同样是解析请求报文体中的查询字符串，根据请求报文中的查询目标尝试从数据库和文件系统中获取需要的数据与图片文件。当数据或图片文件中任一者获取失败时，回调函数 VisualizeCallback() 对用户返回包含查询失败日志信息的 HTTP 响应报文；如果数据与图片文件获取成功，则传给一个对数据和图片文件进行可视化处理的对象，执行包括地理信息解析和数据图像绘制等操作后生成 可视化展示页面，最终将可视化展示页面作为 HTTP 响应返回给用户。回调函数Visualize- Callback() 的伪代码如下：

/\* 引入模块常量 \*/

...

const pool = mysql.createConnection({

socketPath:config.DB\_socketPath,

user : config.DB\_user,

password : config.DB\_password,

database : config.DB\_database});

// 解析Post请求，返回请求中查询的数据与图像

router.post('/',(req,res)=>{

// 解析请求字符串

const queryStr = req.body.queryStr;

pool.query(queryStr,(err, result ,fields) => {

// 查询失败返回失败信息HTTP响应，结束本轮服务

if(err)

{

return res.json({msg : `查询失败，日志信息：\n${err}`});

}

// 查询成功则将数据写入csv文件并与图片打包返回

// 创建输出文件目录：

const outDir = path.join(\_\_dirname,'out',moment().format('yyyy-mm-dd:hh:mm:ss'))

fs.mkdir(outDir);

// 从图片文件目录下复制记录对应的图片文件到输出文件目录：

for(let i = 0; i < result.size; ++i)

{

fs.copyFile(path.join(config.imageDir, result[i].ImageFilename),

path.join(outDir,result[i].ImageFilename),data,

(err) => {

// 图片文件复制失败返回包含文件读写异常信息的HTTP响应，结束本轮服务

if(err)

{

return res.json({msg : `图片文件读取失败，日志信息：\n${err}`});

}

});

}

// 创建csv文件写入流：

const outFileStream = fs.createWriteStream(path.join(outDir, 'out.csv'));

csv\_generate({columns : result, length : result.length}).pipe(outFileStream);

// 压缩文件输出目录

const tarFileName = outDir + '.tgz';

tar.c(

{

gzip: <true|gzip options>,

file: 'my-tarball.tgz'

},

[outDir]

).then(\_ => {

// 在响应中发送打包后的文件

res.send(tarFileName);

}).catch( {err} => {

// 打包失败则发送带有错误信息HTTP响应，结束本轮服务

return res.send({msg : `文件打包失败，日志信息:${err}`});

} );

})

fs.rmdir(outDir,(err)=>{ if(err) console.log('删除临时目录失败') });

});

module.exports = router;

数据可视化展示服务的活动如图[3.11](#_bookmark30)所示。

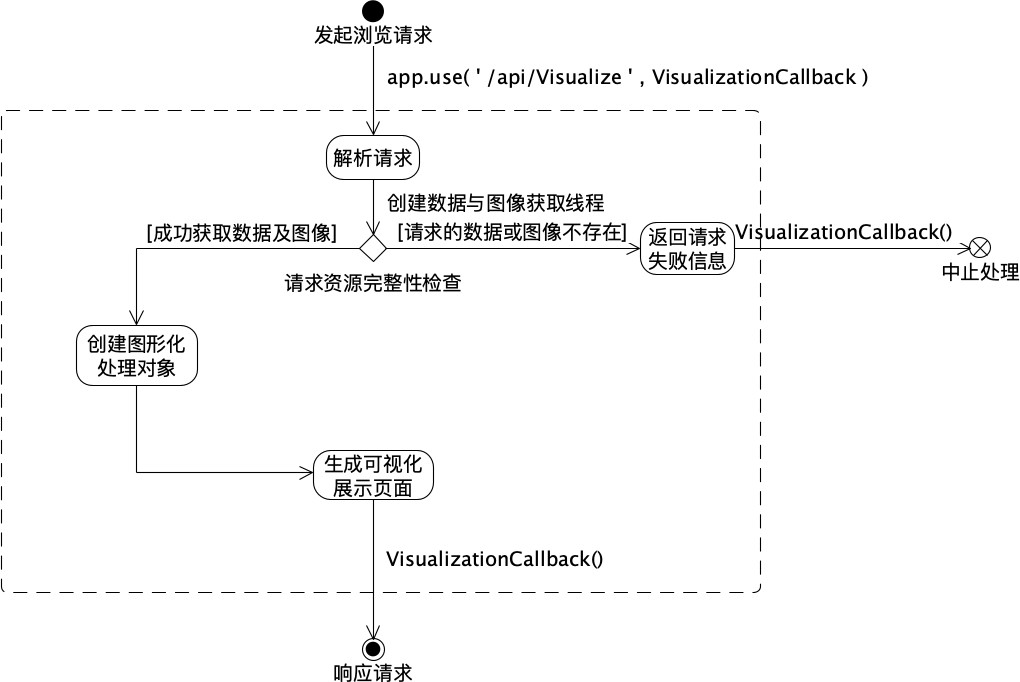


图3.11 数据可视化展示服务活动图

数据可视化展示服务和数据查询服务的活动相比，在中间层会创建接收地理信息和光伏发电数据作为输入的处理对象，完成可视化页面的渲染。

3.发电台站服务设计

发电台站服务包含了图片文件传输服务和数据传输服务两部分，光伏发电台站的主控模块每分钟向服务端发送一次图片文件和数据，每次发送都需要进行具有一定先后次序的确认过程。主控模块首先是在和服务端建立连接后向服务端发送本次测光数据所对应的光伏板表面积灰图片，服务端使用校验码检查到图片已经被完整接收并存储到文件系统中后，再发送接收成功响应并等待主控模块发送数据，否则发送失败响应请求主控模块重发。图片传输成功后主控模块向服务端发送测光数据，服务端再次根据接受到的数据记录中的图片文件名字段检查文件系统中对应图片文件的是否存在，只有监测到图片存在才接收数据并存储到数据库，否则返回错误响应要求主控模块进行图片和数据的重发。

服务端处理图片文件的接收与存储的回调函数是SendImageCallback()，伪代码表示如下：

/\*

函数名： SendImageCllback

输入： 图片文件及对应的MD5校验码

输出： 图片文件的传输结果

\*/

VAR isMD5CheckSumValid : boolean //记录图片文件是否完整无误的布尔值

PROC SendImageCllback(targetImage : 接收到的图片文件描述符(fd),

targetImageMD5 : number,

fs : FileSystemAPI,

ImagePath : 图片文件存储路径(string))

BEGIN

// 生成图片文件MD5码并检查是否一致

imageMD5 <- GernerateMD5(targetImage)

if imageMD5 != targetImageMD5 then return SendImageFailError

// 将图片文件存储到特定目录下，等待接收数据：

fs.copy(targetImage , ImagePath)

send( ImageReceived )

wait( SendDataCallback )

END

处理数据接收与存储的回调函数SendDataCallback()伪代码如下：

/\*

函数名： SendDataCallback

输入： http请求报文中的请求字符串

输出： 日志信息或包含表格文件和图片文件的压缩包

\*/

VAR isImageExist : boolean

PROC SendDataCallback(record : 记录,

conn : 数据库连接对象(DatabaseConnection),

ImagePath : 图片文件存储路径(string),

fs : FileSystemAPI)

BEGIN

// 检查记录对应的图片文件是否存在，不存在则返回错误信息并请求重发

isImageExist <- fs.CheckImageExist(ImagePath, record.imageName)

if not isImageExist then return {ImageNotExistError, ResendRequest}

conn.doInsert( record )

// 返回数据接收成功响应

send(RecordReceived)

END

光伏发电台站使用服务端的发电台站服务发送图片和数据的活动如图[3.12](#_bookmark31)：

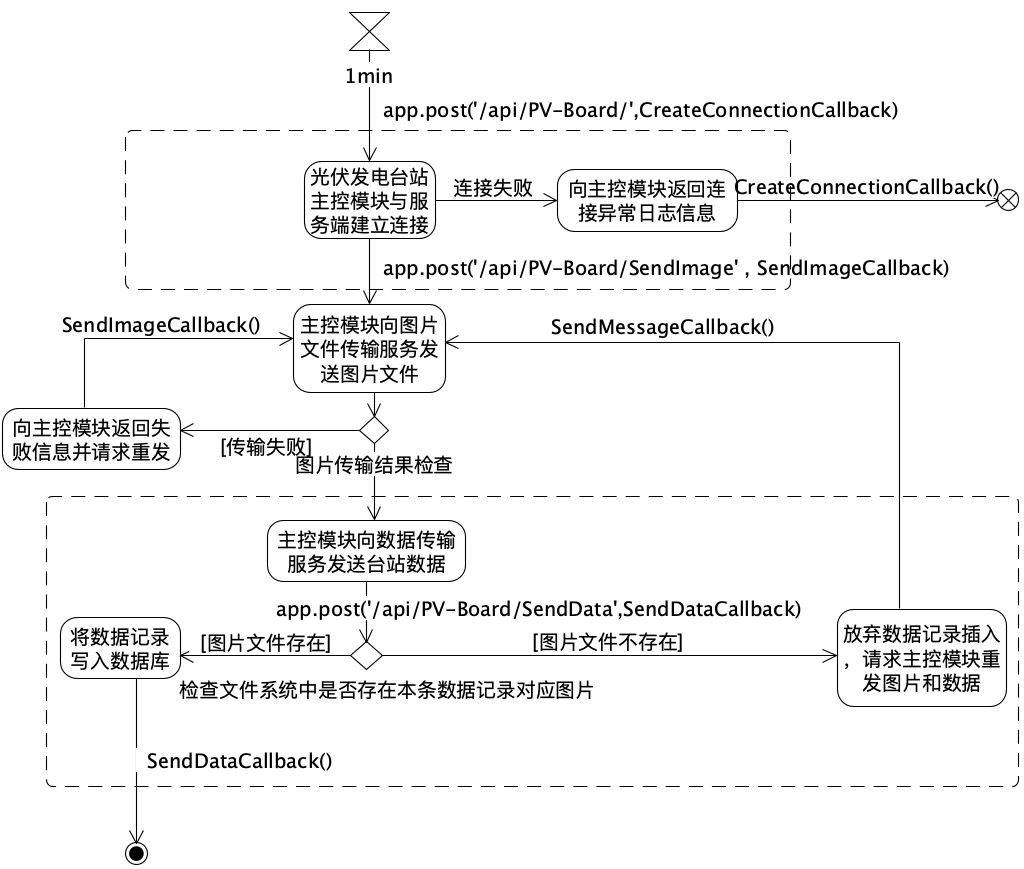


图3.12 发电台站服务活动图

## 3.3 原型系统部署与测试

为了验证本文中服务端软件结构设计的有效性，本项目根据设计方案在CentOS7平台上搭建了包含服务层数据传输模块的原型程序，通过Nginx进行反向代理到80与443端口，可通过接收并解析光伏台站主控模块发送的HTTP请求来实现数据的获取与存储。

光伏发电与气象数据传输模块使用了npm.js提供的Mysql连接器，使用本地套接字的方式连接MariaDB，套接字的路径以及数据库的用户名密码可以在配置文件config.js中修改，而此模块的代码在文件queryRouter.js中，整个服务模块被封装成了一个单独的中间件，加载此模块后数据传输服务被绑定到了对应的路由上，通过监听数据上传者对路由以GET方式或POST方式发送的HTTP请求从MariaDB的线程池中创建线程处理数据插入。创建线程池与处理请求报文的代码节选自附录源文件queryRouter.js：

...

const mysql = require('mysql');

const config = require('./config');

// 读取配置文件并通过本地套接字连接到数据库并创建线程池对象pool

const pool = mysql.createConnection({

socketPath:config.DB\_socketPath,

user : config.DB\_user,

password : config.DB\_password,

database : config.DB\_database});

...

...

//处理请求报文中的数据字段

const i\_StationID = req.body.StationID;

const i\_Time = req.body.Time;

const i\_Power = parseFloat(req.body.Power);

const i\_AvgWindSpeed = parseFloat(req.body.AvgWindSpeed);

const i\_AvgWindAngel = parseFloat(req.body.AvgWindAngel);

const i\_WindSpeedSTDDIFF = parseFloat(req.body.WindSpeedSTDDIFF);

const i\_WindAngelSTDDIFF = parseFloat(req.body.WindAngelSTDDIFF);

const i\_MaxWindSpeed = parseFloat(req.body.MaxWindSpeed);

const i\_MinWindSpeed = parseFloat(req.body.MinWindSpeed);

const i\_Temperature = parseFloat(req.body.Temperature);

const i\_RelativeHumidity = parseFloat(req.body.RelativeHumidity);

const i\_Radiation = parseFloat(req.body.Radiation);

const i\_ReflectRadiation = parseFloat(req.body.ReflectRadiation);

const i\_ComponentTemperature = parseFloat(req.body.ComponentTemperature);

//线程池对象创建新线程执行数据插入，操作结束后回收并销毁此线程

//线程销毁后在回调函数中将结果的日志信息作为HTTP响应返回给数据上传者

pool.query("INSERT INTO PhotoVoltaics value (?,?,?,?,?,?,?,?,?,?,?,?,?,?)",

queryParams,

(err, result, fields) => {

if(err)

{

res.json({msg:"Insert Failed",detail:err});

console.log("Log Shit : ",{msg:"Insert Failed",detail:err});

}

else

{

res.json({msg:"Success",detail:result});

console.log("Log Shit : ",{msg:"Success",detail:result});

}

});

### 3.3.1 GET 方式发送数据测试

图[3.13](#_bookmark34)为使用 Postman 工具模拟主控模块使用 GET 方法进行 HTTP 请求发送。

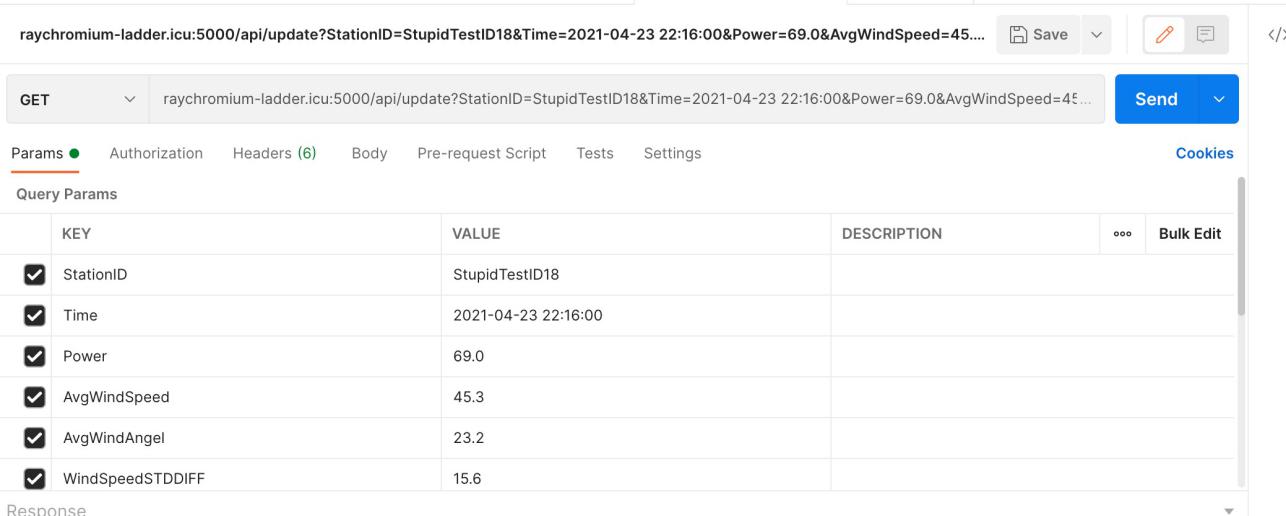


图3.13 模拟GET 方式发送请求

其中使用 GET 方法发送的 URL 中 Qeury 字符串模式如下：

raychromium-ladder.icu:5000/api/update?StationID=StupidTestID18&Time=2021-04-23 22:16:00&Power=69.0&AvgWindSpeed=45.3&AvgWindAngel=23.2&WindSpeedSTDDIFF=15.6&WindAngelSTDDIFF=75.3&MaxWindSpeed=33&MinWindSpeed=69.96&Temperature=39&RelativeHumidity=23.4&Radiation=45&ReflectRadiation=34&ComponentTemperature=50.2

### 3.3.2 POST 方式发送数据测试

图[3.13](#_bookmark34)为使用 Postman 工具模拟主控模块使用 POST 方法进行 HTTP 请求发送。

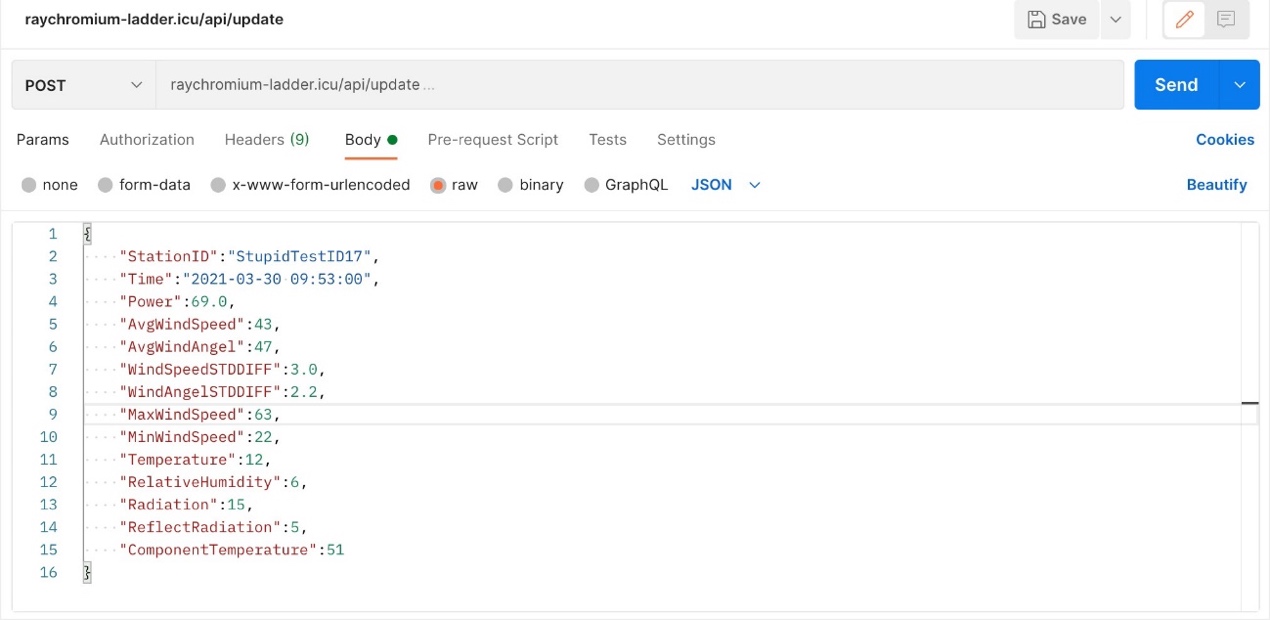


图3.14 模拟Post 方式发送请求

使用POST方法发送的 HTTP 请求头部的Content-Type 字段值为application/json，报文体中携带一个包含待插入数据记录的 JSON 对象，格式如下：

{

"StationID":"StupidTestID17",

"Time":"2021-03-30 09:53:00",

"Power":69.0,

"AvgWindSpeed":43,

"AvgWindAngel":47,

"WindSpeedSTDDIFF":3.0,

"WindAngelSTDDIFF":2.2,

"MaxWindSpeed":63,

"MinWindSpeed":22,

"Temperature":12,

"RelativeHumidity":6,

"Radiation":15,

"ReflectRadiation":5,

"ComponentTemperature":51

}

由服务端进行解析后存入数据库。

### 3.3.3 测试结果

经测试，该服务模块能够高效地接收处理来自多个发起方送达的数据插入请求，并将解析后的数据存入数据库，结果如图[3.15](#_bookmark37)所示。

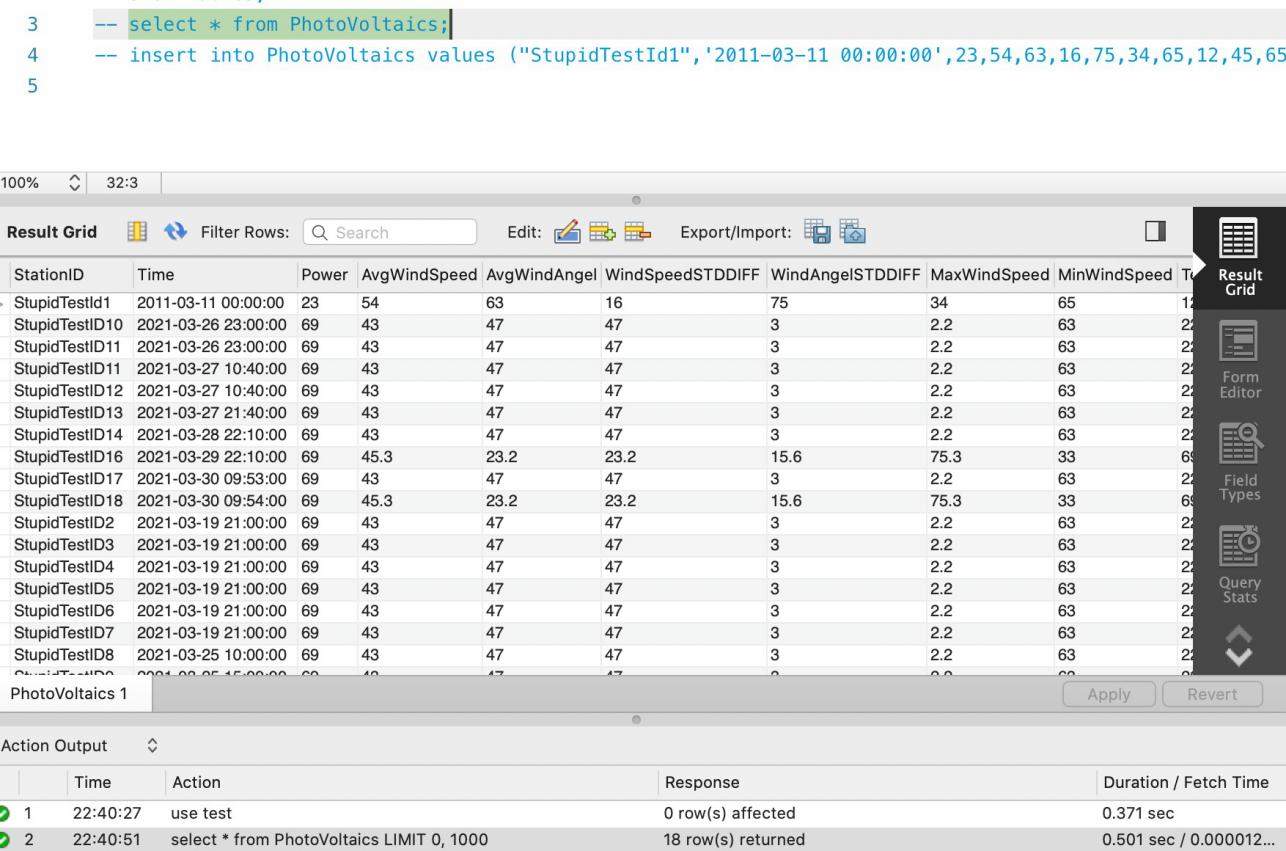


图3.15 数据传输模块测试结果

# 第四章 结论与展望

在光伏发电技术受国家重点投入而得到飞速发展的大环境下，对提高光伏发电效率以及降低损耗的研究成为热点。为了研究光伏板灰尘积累对光伏发电效率的影响，有必要设计一个光伏发电和灰尘监测系统进行实时监测与数据采集。本文着重对光伏板积灰以及发电数据监测系统的服务端进行了分析与设计，构建了一个可拓展的整体应用框架，实现了接收远程传输的光伏发电数据的功能模块并进行了原型系统的部署与测试。

本文的内容可做如下总结：首先根据实验平台以及光伏发电监测系统的实际使用需要，通过用例分析的方式对服务端软件进行了需求分析。

在完成需求分析后，根据各项需求以及模块化开发测试的要求，本项目借鉴了微服务的设计思想，对服务端软件的整体功能架构进行了概要设计。

结合服务端需要具有对高并发请求的处理能力以及事件驱动的特性，本文进行了技术选型，使用Node.JS进行服务端软件原型的开发，在测试平台进行了原型框架的部署，并使用Postman工具模拟多个实际光伏发电台站主控模块发送数据进行了数据传输模块的初步测试。

截至本文完成，实验室中光伏发电台站的气象站部分已基本搭建完成并可获取气象数据。本项目目前仅仅是一个实现了部分功能模块的原型系统，后期将逐步对完善其他功能模块的分析、设计与编码测试的工作，作为灰尘积累数值化分析实验平台的重要组成部分。

# 参考文献

[1] 路甬祥. 清洁、可再生能源利用的回顾与展望 [J]. 科技导报, 2014, 32(Z2):15--26.

[2] 国家能源局. 2020 年上半年光伏发电并网运行情况 [Z]. [EB/OL]. [http://www.nea.gov.cn/](http://www.nea.gov.cn/2020-07/31/c_139254346.htm/) [2020-07/31/c\_139254346.htm/](http://www.nea.gov.cn/2020-07/31/c_139254346.htm/) Accessed April 4, 2021.

[3] 邓红. 家有屋顶太阳能发电站 [J]. 自然与科技, 2013.

[4] 张曦, 康重庆, 张宁, et al. 太阳能光伏发电的中长期随机特性分析[J]. 电力系统自动化, 2014, 38(6):6--13.

[5] 张芳, 邹俊. “十四五”时期我国光伏产业市场培育的前景、困境与路径选择 [J]. 湖北经济学院学报(人文社会科学版), 2021, 18:43--46.

[6] 太阳能电池板捕光及除尘系统 [J]. 实验技术与管理, 2013, (08):36--40.

[7] Said S A, Hassan G, Walwil H M, et al. The effect of environmental factors and dust accumulation on photovoltaic modules and dust-accumulation mitigation strategies[J]. Renewable and Sustainable Energy Reviews, 2018, 82:743--760.

[8] Liu X, Yue S, Lu L, et al. Investigation of the dust scaling behaviour on solar photovoltaic panels[J]. Journal of Cleaner Production, 2021, 295:126391.

[9] 赵明智, 孙浩, 苗一鸣. 沙尘对光伏组件输出特性影响的风洞实验研究 [J]. 动力工程学报, 2019, (9).

[10] 余操, 许盛之, 姚建曦, et al. 灰尘导致的光伏电站发电损失的对比实验 [J]. 太阳能学报, Null. 1--5 .

[11] 莫康信, 李延峰, 王智聪. 光伏电站环境监测系统的设计与实现 [J]. 工程技术研究, 2019, 4(9):4--6.

[12] 周念成, 池源, 王强钢, et al. 光伏并网发电在线监测与状态评估系统 [Z], 2014.

[13] 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, (06):70--73.

[14] 高飞, 何利力, 高金标. 基于 node．js 内存缓存的 web 服务性能研究 [C]. 2015. 109--110.

[15] 赵昆. 改变 web 开发格局的新技术 node.js[J]. 程序员, 2011, (7):124--125.

[16] Simon, McManus. Write maintainable web apps with express[J]. Net, 2014.

[17] 程桂花, 沈炜, 何松林, et al. Node.js 中 express 框架路由机制的研究 [J]. 工业控制计算机, 2016, 29(8):101--102.

[18] Butler H, Daly M, Doyle A, et al. The geojson format[J]. 2016.

[19] 邱祝文. 基于 redis 的分布式缓存系统架构研究 [J]. 网络安全技术与应用, 2014, (10):52--52.

# 附 录

附录中包含了服务端程序 do-query 分支中 0.1 版本的程序代码，可通过处理 HTTP 解析报文中的JSON 获取需要插入的数据记录。

A.1 目录结构

下面是服务端原型程序的目录结构：

├── app.js

├── config.js

├── package-lock.json

├── package.json

├── public

│ └── index.html

└── queryRouter.js

A.2 各文件内容

app.js 是程序的入口点，在事件循环开始前将路由等回调函数推入各阶段的队列中，监听并传入服务器固定端口的 HTTP 请求：

const express = require('express');

const path = require('path');

const fs = require('fs');

const config = require('./config')

const app = express();

// post body json parser middleware

app.use(express.json());

// handle form submition:

app.use(express.urlencoded({extended:false}));

// test basic get request by showing static page:

app.use(express.static(path.join(\_\_dirname,'public')));

// TODO : use queryRouter as the post data update router

app.use('/api/update/',require('./queryRouter'));

app.listen(config.port,() => console.log(`Server Running, Listening on ${config.port}...`));

config.js 以Javascript 对象的形式记录了一些配置信息，包括监听的端口、数据库的连接信息、数据库连接的 socket 路径等：

const { join } = require("path");

const path = require('path');

const config = {

// database config :

DB\_host : "localhost",

DB\_user:"ray",

DB\_password:"2218863",

DB\_database:"test",

DB\_socketPath:"/var/lib/mysql/mysql.sock",

// app config:

port : 5000,

imageDir : path.join(\_\_dirname,'TestImage')

}

module.exports = config;

queryRouter.js 包含从 HTTP 请求报文中解析 JSON 对象或从url query string 中提取数据记录，并从MariaDB 的线程池中创建新的连接进程处理数据插入：

const express = require('express');

const path = require('path');

const fs = require('fs');

const router = express.Router();

const mysql = require('mysql');

const config = require('./config');

const pool = mysql.createConnection({

socketPath:config.DB\_socketPath,

user : config.DB\_user,

password : config.DB\_password,

database : config.DB\_database});

// parse post data and do query

router.post('/',(req,res)=>{

// res.json(req.body);

if(req.body === undefined)

{

return res.status(400).json({msg : "Insert Query Failed : bad json input"});

}

console.log(req.body);

// Check insert attribute values:

// schema :

// StationID,Time,Power,AvgWindSpeed,AvgWindAngel,WindSpeedSTDDIFF,WindAngelSTDDIFF,MaxWindSpeed,MinWindSpeed,Temperature,RelativeHumidity,Radiation,ReflectRadiation,ComponentTemperature

const i\_StationID = req.body.StationID;

const i\_Time = req.body.Time;

const i\_Power = parseFloat(req.body.Power);

const i\_AvgWindSpeed = parseFloat(req.body.AvgWindSpeed);

const i\_AvgWindAngel = parseFloat(req.body.AvgWindAngel);

const i\_WindSpeedSTDDIFF = parseFloat(req.body.WindSpeedSTDDIFF);

const i\_WindAngelSTDDIFF = parseFloat(req.body.WindAngelSTDDIFF);

const i\_MaxWindSpeed = parseFloat(req.body.MaxWindSpeed);

const i\_MinWindSpeed = parseFloat(req.body.MinWindSpeed);

const i\_Temperature = parseFloat(req.body.Temperature);

const i\_RelativeHumidity = parseFloat(req.body.RelativeHumidity);

const i\_Radiation = parseFloat(req.body.Radiation);

const i\_ReflectRadiation = parseFloat(req.body.ReflectRadiation);

const i\_ComponentTemperature = parseFloat(req.body.ComponentTemperature);

const queryParams = [i\_StationID,i\_Time,i\_Power,i\_AvgWindSpeed,i\_AvgWindAngel,i\_AvgWindAngel,i\_WindSpeedSTDDIFF,i\_WindAngelSTDDIFF,i\_MaxWindSpeed,i\_MinWindSpeed,i\_Temperature,i\_RelativeHumidity,i\_Radiation,i\_ReflectRadiation,i\_ComponentTemperature];

console.log(queryParams);

// TODO : Sanity & legal Check

// DO The insert:

pool.query("INSERT INTO PhotoVoltaics value (?,?,?,?,?,?,?,?,?,?,?,?,?,?)",

queryParams,

(err, result, fields) => {

if(err)

{

res.json({msg:"Insert Failed",detail:err});

console.log("Log Shit : ",{msg:"Insert Failed",detail:err});

}

else

{

res.json({msg:"Success",detail:result});

console.log("Log Shit : ",{msg:"Success",detail:result});

}

});

});

// parse get data and do query

router.get('/',(req,res)=>{

// res.json(req.body);

if(req.query === undefined)

{

return res.status(400).json({msg : "Insert Query Failed : bad query url string"});

}

console.log(req.query);

// Check insert attribute values:

// schema :

// StationID,Time,Power,AvgWindSpeed,AvgWindAngel,WindSpeedSTDDIFF,WindAngelSTDDIFF,MaxWindSpeed,MinWindSpeed,Temperature,RelativeHumidity,Radiation,ReflectRadiation,ComponentTemperature

const i\_StationID = req.query.StationID;

const i\_Time = req.query.Time;

const i\_Power = parseFloat(req.query.Power);

const i\_AvgWindSpeed = parseFloat(req.query.AvgWindSpeed);

const i\_AvgWindAngel = parseFloat(req.query.AvgWindAngel);

const i\_WindSpeedSTDDIFF = parseFloat(req.query.WindSpeedSTDDIFF);

const i\_WindAngelSTDDIFF = parseFloat(req.query.WindAngelSTDDIFF);

const i\_MaxWindSpeed = parseFloat(req.query.MaxWindSpeed);

const i\_MinWindSpeed = parseFloat(req.query.MinWindSpeed);

const i\_Temperature = parseFloat(req.query.Temperature);

const i\_RelativeHumidity = parseFloat(req.query.RelativeHumidity);

const i\_Radiation = parseFloat(req.query.Radiation);

const i\_ReflectRadiation = parseFloat(req.query.ReflectRadiation);

const i\_ComponentTemperature = parseFloat(req.query.ComponentTemperature);

const queryParams = [i\_StationID,i\_Time,i\_Power,i\_AvgWindSpeed,i\_AvgWindAngel,i\_AvgWindAngel,i\_WindSpeedSTDDIFF,i\_WindAngelSTDDIFF,i\_MaxWindSpeed,i\_MinWindSpeed,i\_Temperature,i\_RelativeHumidity,i\_Radiation,i\_ReflectRadiation,i\_ComponentTemperature];

console.log(queryParams);

// TODO : Sanity & legal Check

// DO The insert:

pool.query("INSERT INTO PhotoVoltaics value (?,?,?,?,?,?,?,?,?,?,?,?,?,?)",

queryParams,

(err, result, fields) => {

if(err)

{

res.json({msg:"Insert Failed",detail:err});

console.log("Log Shit : ",{msg:"Insert Failed",detail:err});

}

else

{

res.json({msg:"Success",detail:result});

console.log("Log Shit : ",{msg:"Success",detail:result});

}

});

});

module.exports = router;

package.json是本项目npm管理器生成的配置文件，用于记录本项目依赖的npmpackage以及运行脚本，package-lock.json中记录了各个npmpackage的镜像源以及它们所需的依赖，在此不做展示。

# 致 谢

在此衷心感谢指导本项目的张杲峰老师和刘子昂学长，他们在本次毕业设计的过程中从开题、前期准备以及准备过程中的关键时刻为我们提供了耐心的指导与解惑，促进了我们的思考与动手能力的提升。

同样感谢在本次毕业设计题目中与我一起攻克难关的陈宇亮同学，他与设备采购联络人进行了大量的沟通，并且对项目推进做出了良好的分工和时间安排。

最后感谢本科期间兰州大学各位传道授业的老师们，感谢你们把我领进了专业的大门，为我走入工作岗位打下了牢固的能力基础。

本项目由国家电网公司提供资金支持赞助采购设备。

**毕业论文（设计）成绩表**

|  |
| --- |
| **导师评语**  光伏板表面积灰对发电效率影响较大，为了探究其影响程度，需要对光伏板的发电功率及相应的气象等数据进行实时监测与记录，以便后续分析。蔡睿同学的毕业设计针对上述监测和记录，设计并部分实现了用于功率等监测数据上传的服务端软件系统。系统设计基本合理，有一定工作量；论文结构较合理，书写比较规范。论文达到了学士学位的要求，同意答辩，建议授予学士学位。  **建议成绩 中等 指导教师（签字）** |
| **答辩小组意见**  经答辩小组一致讨论，该论文通过答辩，成绩为中等。  **答辩委员会负责人（签字）** |
| **成绩 学院（盖章）**  **年 月 日** |