摘　要

随着现代社会的不断发展，人们越来越多的关注小范围内的环境特征及环境变化。为满足对小范围内环境监测的需要，本文设计并开发了一套微环境监测平台。针对平台高速缓存、高并发量、读写分离以及实时性等性能要求，本文面向微环境监测平台设计开发了一套存储系统。本文针对微环境监测平台与海量数据存储之间的问题进行研究，研究内容大致分为以下三点：

（1）　高并发数据接口设计　本文设计高并发数据接口并实现服务端程序，通过采用线程池技术，以合理的线程数达到最大的数据处理量，并利用 Http 接口，完成网关与服务端之间数据传输与接收工作。

（2）　基于读写分离的数据库存储方案设计与优化　根据平台不同业务数据对存储方面的不同要求，本文分别设计了增加读库和数据缓存两种读写分离的存储方式，其中增加读库的方式是采用 MongoDB 数据库作为主库，多个 MySQL 数据库作为读库；数据缓存方法是采用 Redis 数据库作为内存数据库实现缓存，实现平台的高并发与实时性的性能要求。另外本文根据数据库存储数据的不同特点分别对 MySQL 数据库和 MongoDB 数据库设计不同的优化方案，提高数据库的利用率，减少资源浪费。

（3）　数据存储中间件的设计与实现　结合微环境监测平台数据存储的需求分析以及平台的具体业务，提出一套适合本平台业务的数据存储中间件的设计方案，并从垂直分片和水平分片这两个方面对该方案进行详细设计，最大程度的提高微环境监测平台的查询效率。

实验结果表明，当平台存取海量数据时，整个平台运行稳定，存储效率高，数据存取正常，能够保证平台数据的实时性，本文所设计的存储方案具有一定的可行性与有效性。同时对本研究进行了总结，结合实际需求，提出了平台设计的改进思路，为课题今后进一步深入研究提供参考。

关键词　微环境监测；读写分离；MySQL；Redis；MongoDB；中间件

Abstract

With the continuous development of modern society, people pay more and more attention to the environmental characteristics and changes in a small range. In order to meet the needs of small-scale environmental monitoring, this thesis designs and develops a micro-environment monitoring platform. Aiming at the performance requirements of platform cache, high concurrency, read-write separation and real-time, this thesis designs and develops a storage system for micro-environment monitoring platform. In this thesis, the problems between micro-environment monitoring platform and mass data storage are studied. The research contents can be divided into three parts:

(1) Design of high concurrency data interface. We design a high concurrent data interface and implements the server program. By using thread pool technology, the maximum data processing capacity can be achieved with a reasonable number of threads, and Http interface is used to complete the data transmission between gateway and server.

(2) Design and optimization of database storage scheme based on read-write separation. According to the different requirements of different business data on different platforms, we design two separate storage modes of adding read-write database and data cache, in which the way of adding read-write database is to use MongoDB database as the main database and multiple MySQL databases as the main database. As a read library, the data caching method is to use Redis database as a memory database to achieve caching, and achieve high concurrency and real-time performance requirements of the platform. In addition, according to the different characteristics of the database storage data, this paper designs different optimization schemes for MySQL database and MonoDB database, to improve the utilization of the database and reduce the waste of resources.

(3) Design and implementation of data storage Middleware. Combining with the requirement analysis of data storage in micro-environment monitoring platform and the specific business of the platform, a design scheme of data storage middleware suitable for this platform is put forward, and the scheme is designed in detail from two aspects of vertical slicing and horizontal slicing. This scheme improves the query efficiency of the micro-environment monitoring platform to the greatest extent.

The experiment results show that when the platform accesses massive data, the whole platform runs stably, the storage efficiency is high, the data access is normal, and the

real-time performance of the platform data can be guaranteed. The storage scheme designed in this thesis is feasible and effective. At the same time, this thesis summarizes the research, and puts forward the improvement ideas of platform design combined with the actual needs, which provides a reference for further in-depth study of the subject in the future.

Key words Microenvironment Monitoring; Read and Write Separation; MySQL; Redis; MongoDB; Middleware

目　录

第 1 章　绪　论

1.1　研究背景和意义

现阶段人们所熟知的气象预报和环境监测主要是对大范围内的环境数值进行观测，但是在日常的工作生活中，小范围内的环境变化更需要多加关注，因此微环境监测显得尤为重要。微环境指的是一个范围较小的特定区域的环境，通过对该区域的环境进行监测，监测得到的环境数据可以为生活生产提供辅助决策。目前微环境监测已引起国内外科学家的关注。

随着现代社会科技的不断发展，各个行业也发展迅速，数据量呈指数增长，数据的增长速度也越来越快，传统的数据库已经不能满足日益增长的庞大数据量的存储需要[1]。另一方面，目前各种类型的传感器、定位系统、监控器等的使用也变得很普遍，这些类型的数据源同样也产生了海量数据，新型的数据源给传统的存储方式带来了新的机遇与挑战[2]。传感器数据通常以秒为单位进行数据的采集，通常情况下采用传统的数据库进行数据的存储会造成极大的数据冗余，传统的数据库已经不能满足海量数据的存储、分析和管理的需要。因此，寻找一种高效的传感器数据存储方案成为数据库应用的研究热点[3]。

本文来源于河北省重点研发项目《基于语义物联网的微环境监测平台关键技术研究》。基于语义网的物联网微环境监测平台针对物联网的三个层次进行研究和设计，提供标准的接口设备和通信协议。基本平台完成后，可以在本平台上进行不同的业务应用。微环境监测仪器可在各种无人值守恶劣环境下长期稳定的进行数据的采集工作。传感器设备能够根据不同的需求选择要进行监测的内容。同时，可以通过采用统一通信协议的方式保证数据传输的过程中的的通信质量。最终将采集到的数据以及项目基础信息等集中到一个平台中，进行统一的存储和处理，为平台的用户进行数据的直观展示，并进一步提高工作效率。

本文为项目提供一个能够满足高速缓存、高并发量、读写分离以及实时性等性能要求的存储系统以及一套微环境监测平台。该平台通过对传感器采集的空气温、湿度，土壤温、湿度、光照强度和 CO2 等多种环境数据进行存储、管理、分布和展现。通过对平台进行需求分析，平台采用 Java 语言进行开发，业务架构设计采用 SOA 架构，并利用 layer 框架实现局部刷新。同时平台采用 GIS 技术实现对传感器数据的地图显示、管理等功能。另外，为实现平台的高并发性能要求，本文基于线程池技术设计了高并发数据接口。利用该接口能够实现大数据量时，网关与服务端之间数据的顺利传输与接收工作。根据平台不同业务数据对存储方面的不同要求，本文分

别设计了增加读库和数据缓存两种读写分离的存储方案来实现对数据的存储工作。同时本文根据数据库存储数据的不同特点分别对 MySQL 数据库和 MongoDB 数据库设计不同的优化方案，提高数据库的利用率，减少资源浪费。另外，本文结合微环境监测平台数据存储的需求分析以及平台的具体业务，提出一套适合本平台业务的数据存储中间件的设计方案，并从垂直分片和水平分片这两个方面对该方案进行详细设计，最大程度的提高微环境监测平台的查询效率。

1.2 国内外研究现状

我国的环境监测工作在上个世纪 70 年代就已经开始发展了。到目前为止我国在环境监测方面的能力已经有了非常明显的提高，逐渐的形成了以环境监测站为中心的监测体系[4]。它已经有组织网络的雏形，并且监测分析技术也形成体系化。

我国的学者专家等对环境监测方面进行了深入的研究。张志君[5]等改进了传感器技术在大系统环境监测中的实际应用。同时基于对智能环境的监控管理技术，提出了 IPMI 智能接口标准，在很大程度上对环境监测技术进行了优化，并对系统的稳定性以及安全性进行了大幅度的提高。另一方面，基于 ZigBee 技术的环境监测技术已成为我国的研究热点问题[6]。陈亚楠[7] 根据采用大数据量的由卫星传感器主要组成的环境监测网络，设计出利用 ZigBee 为主要技术手段的环境监测系统，主要功能包括长时间内连续进行数据的采集、上传和控制等功能[8]。

另一方面，从 19 世纪末开始国外就已经开始对环境监测技术进行研究[9]。首先由发达国家开始着手进行研究，最初在全国范围内监测的目标包括空气、水以及放射性污染物等[10]。随着对环境监测体系的不断研究和完善，环境监测平台的发展形式变得更加多样化、标准化。根据卫星通讯系统成立的水体污染的监测项目，主要是由西欧国家一起相互合作完成。此项目主要是对多瑙河的水质进行监控。日本根据长时间连续的对监测设备进行在线测量并处理数据，成立了一些监测水、空气等环境指标的监测平台[11]。在 2004 年，美国在环境监测方面的研究员利用无线传感网络通过对葡萄园中的生长环境进行了大范围的监测，成功的提升了葡萄的品质。

随着云计算、大数据等技术的不断发展，传统的存储方式已经不能满足海量数据存储的需要。非关系型的数据库发展迅速，目前在该领域中已变得炙手可热[12]。现阶段最具代表性的 NoSQL 有 Hbase、Redis、MongoDB、Neo4j 等。本文选用 Redis 和 MongoDB 这两种非关系型数据库并结合 MySQL 关系型数据库设计了一套面向微环境监测平台的存储系统。Redis 和 MongoDB 两种非关系型数据库发展迅速，应用广泛。

阿里云和腾讯等大型网络公司都应用了 Redis 技术。阿里云的 Redis 版云数据库主要采用了能够实现无缝扩展的集群架构和双机热备架构[13]，具有较高的稳定性，

同时能够保证数据的安全可靠。另外该云数据库能够实现故障自动迁移，修复安全漏洞，大大的提高了服务的稳定安全的性能。腾讯云的云存储 Redis 主要提供了集群版和主从版的存储服务以及兼容 Redis 协议的缓存服务[14]。同时它可以支持主从式热备用，提供自动灾难恢复切换、数据备份、故障迁移、实例监控、在线扩展、数据备份等完整的数据库服务[15]。

MongoDB 技术也在阿里、网易等大型公司中得到了大范围的应用。阿里云的 MongoDB 版的云数据库能够提供多种类型的功能，如时间点备份以及安全审计等等[16]。目前它已经广泛应用于游戏、金融、互联网等众多领域[17]。另外，MongoDB 云数据库支持弹性扩展，提供高可靠性的存储引擎。网易云的 MongoDB 则是能够支持系统中出现故障自动切换的服务[18]。当系统中主节点出现故障的情况，它能都在最短的时间内完成故障的转移工作。采用性能较高的架构，专业的数据监测控制，使运维工作变得更为简单便捷[19]。

1.3 研究内容

本文来源于河北省重点研发项目《基于语义物联网的微环境监测平台关键技术研究》。本论文旨在为项目提供一套微环境监测平台以及面向该平台设计一个高性能的存储系统。本平台通过对传感器采集的空气温、湿度，土壤温、湿度、光照强度和 CO2 浓度等多种环境数据进行存储、管理、分布和展现。同时为满足微环境监测数据的时效性、高并发等需求，设计一套存储系统。其中主要功能包括高并发数据接口设计、基于读写分离的数据库存储方案设计与优化、数据存储中间件的设计等方面，力求实现平台数据展示的实时性。

本文的研究内容大致分为以下三点。

(1) 高并发数据接口设计 为解决大数据量时平台无法同时接受数据的问题，本文通过采用线程池的技术，利用 Http 接口，设计并实现了高并发数据接口。通过该接口实现在大数据量时网关与平台之间顺畅的数据传输与接收工作。

(2) 基于读写分离的数据库存储方案设计与优化 根据平台不同业务数据对存储方面的不同要求，本文分别设计了增加读库和数据缓存两种读写分离的存储方式来实现对海量数据的实时存储。另外本文根据数据库存储数据的不同特点分别对 MySQL 数据库和 MongoDB 数据库设计不同的优化方案，提高数据库的利用率，减少资源浪费。

(3) 数据存储中间件的设计与实现 结合微环境监测平台数据存储的需求分析以及平台的具体业务，提出一套适合本平台业务的数据存储中间件的设计方案，并从垂直分片和水平分片这两个方面对该方案进行详细设计，最大程度的提高微环境监测平台的查询效率。

1.4 论文的组织结构

本论文的结构安排如下。

第1章 绪论。首先对当前科技发展情况下微环境监测平台的存在意义进行了介绍，分析了目前传统的存储系统存在的问题，并进一步提出面向微环境监测平台设计一套存储系统来解决这些难题。后面对环境监测、存储技术以及中间件的国内外研究现状进行了简单介绍，并简单的阐述了本文的研究内容以及作者的主要工作。最后介绍了一下本篇论文的组织结构。

第2章 微环境监测平台设计。本章通过对微环境监测平台的分析，以现阶段需求为基础，分别对平台的业务架构设计、平台功能设计、数据存储设计进行了介绍，并利用平台主界面对平台的设计结果进行了可视化展示。

第 3 章 高并发数据接口设计。为解决大数据量时平台无法同时接受数据的问题，本文通过采用线程池的技术，利用 Http 接口，设计并实现了高并发数据接口。通过该接口实现在大数据量时网关与平台之间顺畅的数据传输与接收工作。

第 4 章 基于读写分离的数据库存储方案设计与优化。根根据平台不同业务数据对存储方面的不同要求，本文分别设计了增加读库和数据缓存两种读写分离的存储方式来实现对海量数据的存储。另外本文根据数据库存储数据的不同特点分别针对 MySQL 数据库和 MongoDB 数据库设计了不同的优化方案，提高数据库的利用率，减少资源浪费。

第 5 章 数据存储中间件的设计与实现。本章结合微环境监测平台的项目背景，设计了一款数据存储中间件。该中间件主要是从数据分片的角度进行设计的，其中包括垂直分片和水平分片两种分片方式。最后使用数据查询的方式对数据中间件的性能进行了测试验证。结果表明该数据存储中间件具有一定的可行性。

第 6 章 工作总结与下一步规划建议。对本文所设计的微环境监测平台存储系统的优缺点进行了总结，同时对下一步工作规划提出建议。

第 2 章　微环境监测平台设计

2.1　需求分析

为了满足人们对于小范围区域内环境监测的需求，解决环境数据无法系统分析的问题，同时也为了方便相关工作人员对监测到的环境数据进行管理，本文设计了一套微环境监测平台。

该平台主要是对传感器采集的多种环境数据进行接收、存储、展示以及管理等处理。传感器需要采集的环境数据主要包括 CO2 浓度、光照强度以及土壤的温、湿度和空气的温、湿度等多种信息。平台的主要包括传感器节点的地图展示、用户信息管理、项目信息管理、区域管理、节点管理、设备管理、信息查询管理以及预警管理等功能。微环境监测平台整体架构图如图 2-1 所示。

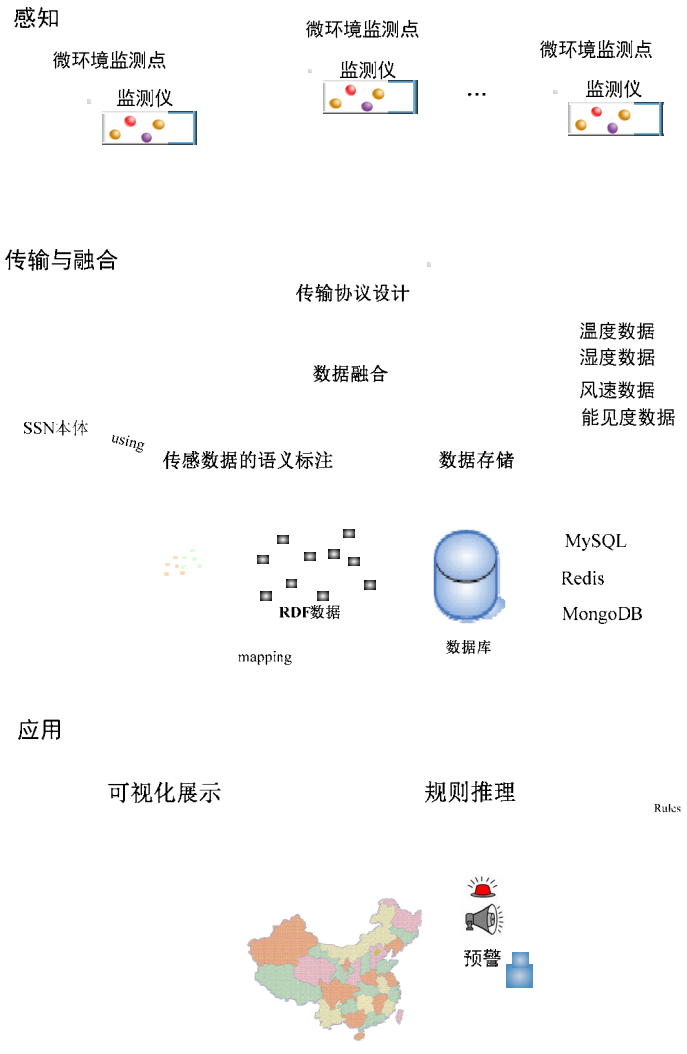

图 2-1　微环境监测平台整体架构图

2.2 平台业务架构设计

本文在设计了微环境监测平台的同时又针对该平台设计了一套高性能的存储系统。该存储系统开发完成后可以使平台在处理大数据量时，能够实现平台的高并发量以及高速缓存，保证数据的实时性以及稳定性。根据平台不同业务数据对存储方面的不同要求，本文分别采用MySQL数据库、MongoDB数据库以及Redis内存数据库三种数据库，并设计了增加读库和数据缓存两种读写分离的存储方案来实现对数据的存储工作。同时根据数据库的不同特点分别对MySQL数据库和MongoDB数据库设计不同的优化方案。另一方面，系统采用数据分片的方式设计了一套数据存储中间件的设计方案。本平台的技术路线图如图2-2所示。

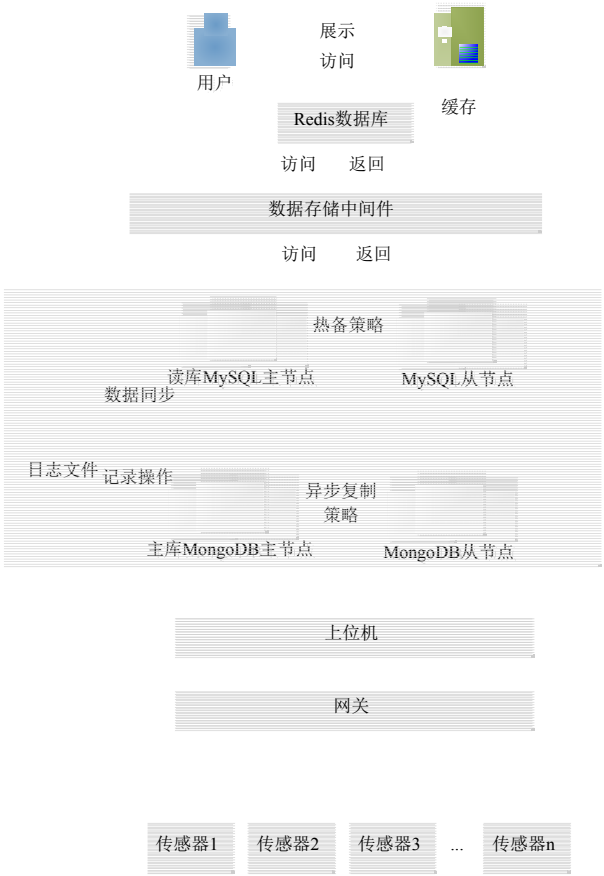

图 2-2　微环境监测平台技术路线图

微环境监测平台主要是通过以传感器设备采集的数据为依据来实现作物的生长环境进行监测。本平台适用于林业、牧业、道路、水质监测等各个行业中，可以实现同时对数个项目的同时创建。因此，为保证同一类型项目实现结构的共享并保留自身所特有的特点，本平台在服务端利用 SOA 架构进行业务的设计。服务端 SOA 的架构图如 2-3 所示。

SOA 架构指的是一种接口清晰、结构灵活的体系结构。该架构能够实现将业务中的分散的多种类型的功能按照一定的条件进行提取，并将提取之后的功能按照一定的规则进行重组。本平台使用该架构后，可以在一定程度上减轻了开发人员的重复开发的压力。同时有利于开发人员可以在尽可能短的的时间内完成程序的开发工作。

图 2-3　服务端 SOA 架构

2.3　平台功能模块设计

微环境监测平台是通过采用传感器设备对作物的成长环境进行监测，将采集到的环境数据展示到平台上，微环境监测者提供直观的数据变化，有利于平台用户对数据进行分析总结。通常本平台传感器经常采集的环境数据主要包括 CO2 浓度、光照强度以及土壤的温、湿度和空气的温、湿度等多种信息。平台的主要包括传感器

节点的地图展示、用户信息管理、项目信息管理、区域管理、节点管理、设备管理、信息查询管理以及预警管理等功能。微环境监测平台的功能模块图如图 2-4。

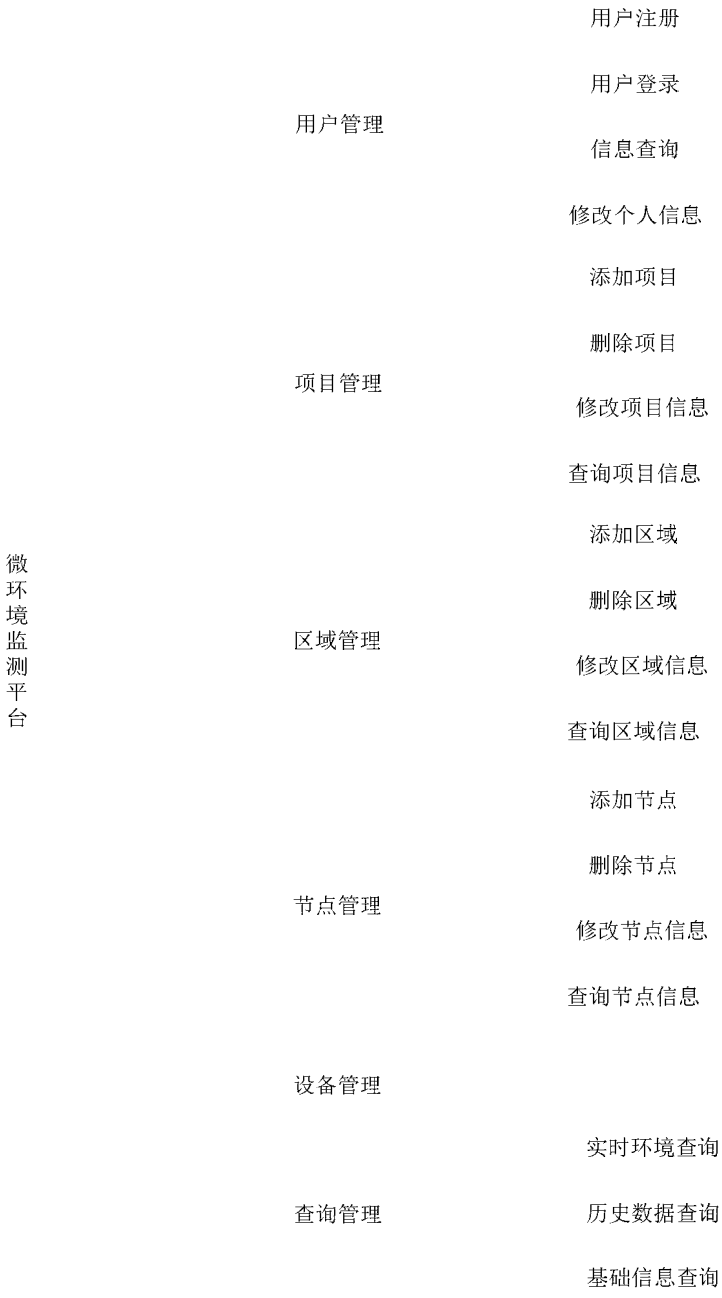

图 2-4　微环境监测平台功能模块图

在本平台中，用户管理模块主要包括用户注册和登陆，还有信息查询以及修改个人信息等功能。当用户注册信息成功后，用户可以通过使用自己注册的用户名以及与用户名相匹配的密码完成登陆，进入平台处理相关业务。项目管理模块可以实现的功能主要有添加项目、删除项目、修改项目信息以及查询项目信息等等。用户能够根据项目需求，对项目进行以上各种操作从而达到自己的目的。另外，平台中其他模块，如区域管理模块、节点管理模块以及设备管理模块。这些模块的相关功

能和项目管理模块的功能相类似。

查询管理模块的主要功能有实时环境查询、历史数据查询以及基础信息查询。实时的环境数据为最近的一个时间段内传感器采集的数据。超过这个时间段内所有的环境数据都属于历史数据。另外基础数据指的是平台中用户、项目以及设备等的相关信息。

2.4 数据存储总体设计

图 2-5　平台 E-R 图

为了能够完整清楚的呈现出平台中数据库各种信息的属性关系，该系统的 E-R 图如图 2-5 所示。

根据本平台的平台需求，分别对平台中数据的属性和相互之间的联系进行分析，设计的数据库表包括用户表、项目表、区域表、节点表、传感器表以及数据表。

其中，用户表主要包括用户 id、用户名、密码、电话、注册时间以及用户照片等信息。项目表中主要包含的信息有编号、项目名称、简介、创建时间、访问网站、负责人编号、以及区域编号等。区域表中信息主要有区域编号、类型、名称、区域照片、区域描述、布控时间、区域状态以及该区域所对应的项目编号。节点信息主要包括编号、节点名称、节点照片、节点描述、添加时间、经纬度、节点状态和节点所隶属的区域编号。传感器设备的所包含的信息为编号、设备名称、设备类型、

设备照片、设备描述、设备的厂商编号、添加时间、以及设备对应的节点编号、获取数据的时间和获取数据等等。数据表中的信息主要包括编号、名称、数值、时间、传感器编号。

另一方面，按照各数据表的属性以及各表的表间关系可以看出，用户使用平台时查看具体项目的权限是由用户具体负责哪个项目决定的。用户可以负责多个项目，每个项目又可以包含多个环境监测的区域，其中每个区域又由多个节点组成，一个节点同时又包含多个传感器设备。

2.5 微环境监测平台的主界面

通过对微环境监测平台的需求分析，本平台主要从功能设计、业务架构设计、数据存储设计等几方面进行设计。平台的主界面如图 2-6 所示。

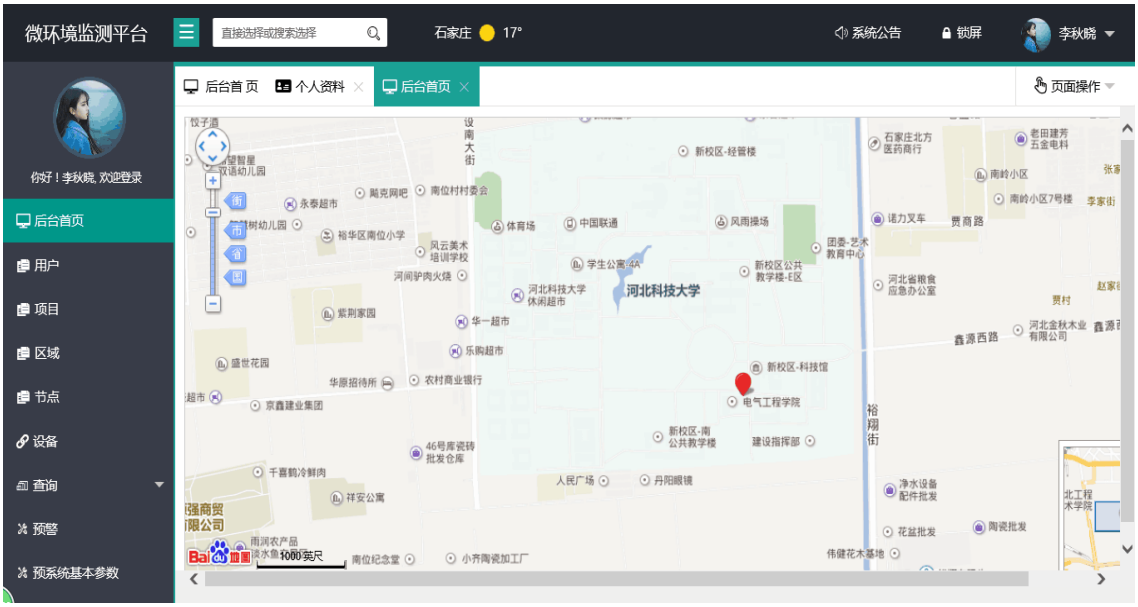

图 2-6　微环境监测平台主界面

从平台的主界面中可以看出，微环境监测平台包括用户管理、项目管理、区域管理、节点管理、设备管理、信息查询管理以及预警管理七大功能模块。通过地图的形式展示数据，使用户查看数据更为直观，更容易理解，为使用该平台的环境监测人员带来方便。

2.6 本章小结

本章首先对微环境监测平台进行了分析，以现阶段需求为基础，对系统整体进行设计，然后分别对平台的业务架构设计、平台功能设计、数据存储设计以及平台主界面的展示几方面进行了介绍，为后面章节中面向平台的存储系统做好铺垫。

第 3 章　高并发数据接口的设计与实现

本章主要介绍系统中高并发数据接口的设计与实现，主要使用线程池技术，通过使用合适的线程个数来达到最大的数据处理量，并通过 Http 接口完成网关与服务端的数据接收和传输任务，保证存储系统可以有较强的负载均衡能力以及高并发能力。

3.1　需求分析

本论文中基于微环境监测平台的存储系统是可以应对大量的监测数据的。首先，一般情况下在使用传感器对环境进行监测，并采集检测数据时，传感器以较短的时间间隔进行数据采集，在传感器较多时，短时间内，如几小时或者几分钟内就会产生大量的环境数据；其次，当用户访问量变大时，系统平台的资源利用率会有一定幅度的升高，达到设定当阈值后，系统性能便会有下降。所以在海量的并发请求发生后，服务器要保证流畅的服务和稳定的用户体验，要达到该目标，系统设计就要有应对高并发的能力，这就需要有优秀的通信服务架构。除此之外，为了避免大数据量时系统会出现长时间的延时，从而导致平台出现数据冗余的问题，提高平台的数据存储读取性能。本文在高并发的数据接口中使用了线程池技术。

因此，本文中系统的存储部分要具有高并发能力和负载均衡能力，因此本章对高并发数据接口和线程池进行较为深入的研究。

3.2　关键技术

3.2.1　高并发技术

该系统的重要性能指标就是是否具有高并发能力，它要求平台可以在短时间内对大量的请求任务作出相应[20]。本文设计的存储系统是面向具有多传感器的微环境监测平台，环境数据除了数据量和类型多之外，还要求系统具有实时性和丢包率低的特点，传感器与系统之采用高并发长时间连接，因此平台必须具备应对高并发量的能力[21]。如果系统不能保证较高的并发量，环境数据的交互将会出现较大的延时，甚至会吃西安无法正常传输，影响系统数据的实时更新和性能[22]。

3.2.2　线程池技术

为了避免因数据处理不及时从而导致平台中出现大量的数据冗余问题，并提高平台的性能。本研究在高并发数据接口中引入了线程池技术。线程池是一种高级的多线程处理形式[23]。它并不是简单的将多线程进行罗列后再使用，而是按照一定的处理规则对任务进行处理。线程池技术可以有效的降低多线程任务中的资源消耗，

提高系统处理能力[24]。所以线程池技术在服务器程序中大量被使用，其可以最大程度的利用系统的资源，降低系统开销[25]。

线程池的工作过程如图3-1所示。当客户端与服务器连接成功后，首先系统会将任务添加到系统队列中，然后通过对客户端连接请求进行监听，判断客户端是否有请求出现[26]。如果客户端有请求出现，则将请求添加到任务队列。任务队列指的是在高并发的情况下任务数与线程池可以处理的数目有一定差距的时候，可以用来进行缓冲的队列。

如果客户端没有请求出现，则返回上一步，对客户端继续进行监听。当客户端出现请求并把请求添加到任务队列之后，然后需要判断线程池中是否有空闲线程[27]。如果线程池中存在空闲线程，则接受客户端发出的请求，并与客户端进行通信。线程池接收到客户端请求后，则开始对文件进行相关处理操作[28]。当对相关文件进行处理之后，工作线程就将进入睡眠状态。等到下次出现新的请求再将线程唤醒，对请求进行处理。如果没有空闲线程，客户端的请求需要继续在任务队列中继续等待[29]。等到有空闲线程出现后，再对请求进行处理。

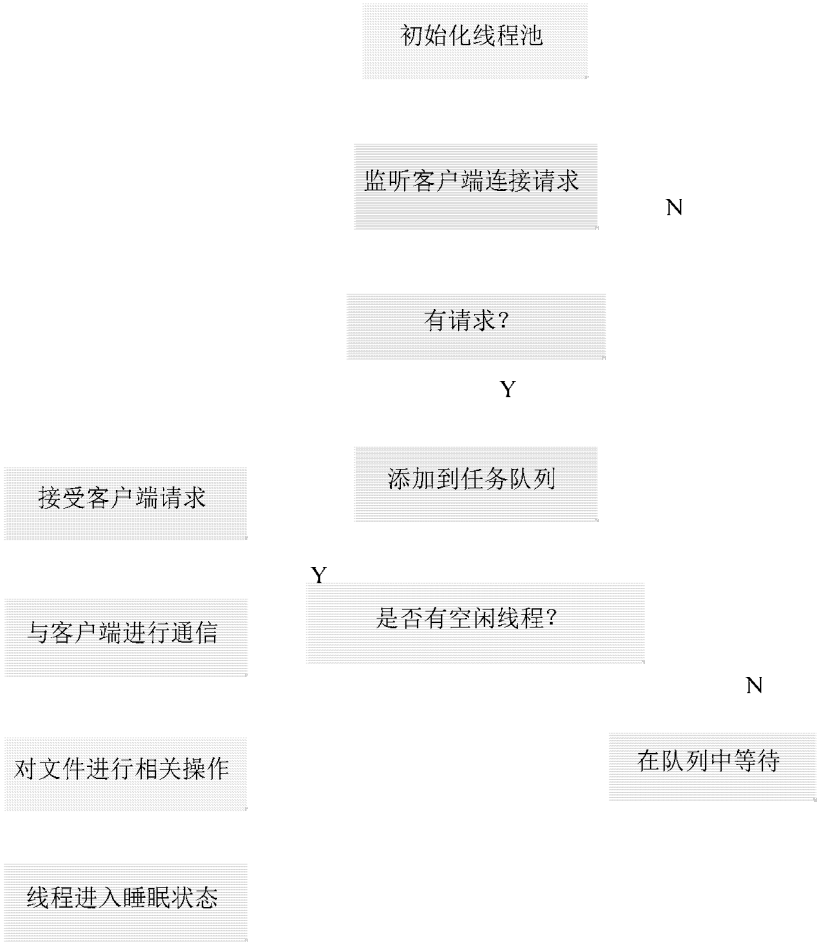

图 3-1　线程池工作流程示意图

3.3 高并发数据接口的设计

本论文针对微环境监测平台所设计的高并发数据接口，主要对传感器采集的数据以及设备状态信息进行处理，数据的传输主要有以下几个步骤。

首先将传感器部署到需要监测的环境中，然后通过传感器对环境中要监测的数据进行采集，如空气温、湿度，土壤温、湿度，光照强度等。数据采集好之后需要将这些数据上传到网关，由网关对数据进行接收。接下来网关将数据上传到上位机中，然后通过上位机将数据传送至服务器端。在服务器端由高并发数据接口对大量的传感器数据进行接收工作。在此过程中接收器会根据本地时钟为该数据产生一个用来判断传感层数据实时性的时间戳。

当平台中短时间内同时传输与接收大量的数据时，平台可能不能在第一时间内对数据进行处理，因此可能会造成大量的数据堆积现象。为防止这类情况的发生，本文在高并发数据接口中引入线程池技术。该技术能够有效的降低对数据的复制操作次数，大幅度的提高平台的性能。其过程如图 3-2 所示。

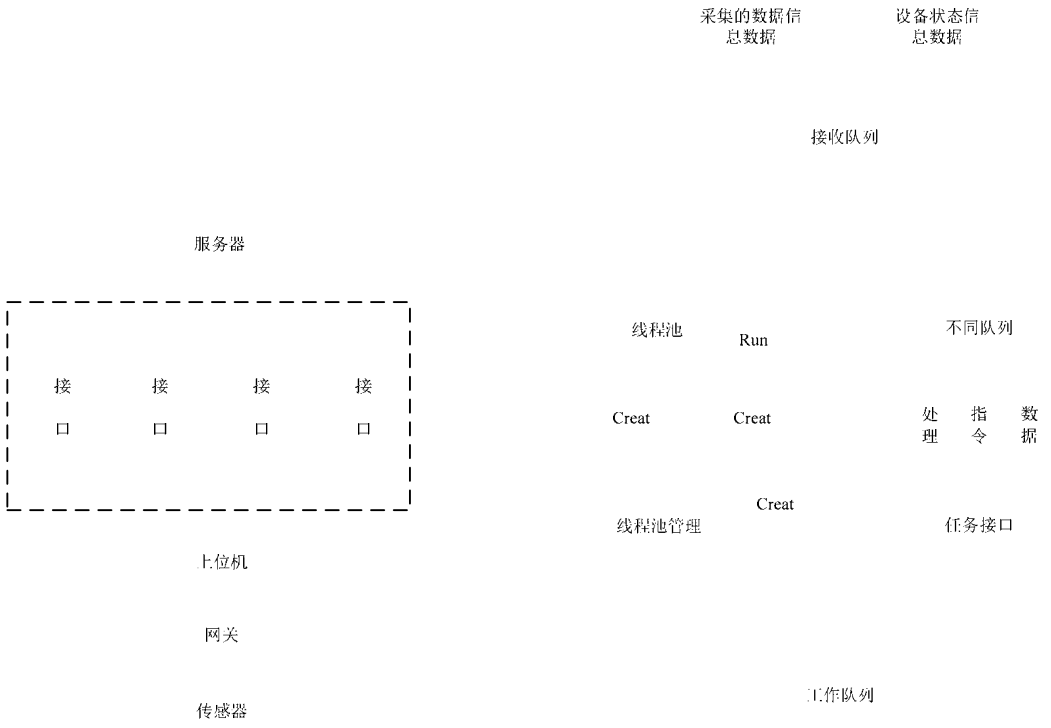

图 3-2　高并发数据接口

本平台利用对线程池技术来处理微环境监测平台中上传的大量数据，以及接收上层下达指令。在数据接收过程中，系统通过线程池中的接收线程，接收获取设备的状态数据和环境数据，并将其存入接收队列。然后线程池中的工作线程将会从接收队列中按先后顺序取出数据并处理，处理后的结果将为存储入数据库。工作线程将从上层服务端接收的指令存储入指令队列，下发线程将从指令队列中取出指令逐

个下发给传感层的各个采集点。

任务队列中存放着一系列平台中的相关任务。其中任务队列中包括任务的名称、类型、参数以及任务本身。任务队列除了可以将存储任务外，还要提供相应的接口。常见的接口一般包括：查看当前队列的执行状况，删除还没有处理的任务以及清空队列。任务队列通常情况下都是按照任务的优先级来处理相应的任务。队列会对当前的任务状态进行判定。只有在任务处于完成状态，或者出现错误或超出一定的时间后，队列才会开始执行下一个任务，直到队列为空。

如果想对系统中不同队列的数据实施操作，可以采用新建任务接口和线程池的方法进行处理。其中，处理队列中所有的任务都要使用任务接口。当线程池中有任务需要要执行时，就要调用任务接口。任务接口与平台实际任务没有任何关系，相互独立，两者之间不会产生任何影响。根据平台的业务分析得出，本文可以抽象出的接口有设备状态信息解析任务接口和采集数据解析任务接口，还包括控制指令封装接口。工作中实际调用任务接口处理任务的线程称作是工作线程。

为了确保传输层中数据的实时性，除了对通信网络进行优化外，还需要提高系统平台的并发量。因此，针对微环境监测平台服务端面对底层大数据量带来的压力，进行了高并发数据接口设计，并结合平台需求实现了服务端高并发数据接收程序。该数据接收程序主要是针对数据量大时，传统的程序不能在短时间内对大量的数据进行接收，很可能会造成数据的大量冗余，从而导致服务器出现宕机，影响平台的使用，降低工作人员的工作效率。针对高并发数据接收程序，本文设计了一套高并发数据接收程序算法，主要是用来处理线程池中具体业务。该算法的具体设计如表3-1所示。

表 3-1 服务端高并发数据接收程序算法

Input	Device status information || Collected data information || Issued instructions
Process	1：通过 Thread_pool 提供锁机制、工作线程、任务队列等引用，Thread_worker 提供任务接口，poolinit()指定线程数 2：上位机上传数据,服务端 DataReceive()接收数据，调用 PoolAddWorker()添加数据入任务队列 3：线程池分派线程处理数据，调用 Thread_taskCall() 4: if 空闲线程 = null && 现有线程数 < MaxThreadSize：开启新线程，解析数据 data 5：if 包头 = AA00 and 包尾 = A00A：DataInforAnalysis()←DataInformation Protocol 6： if 队列 1 被占用（lock）：等待， else：入队列 1（数据信息队列） 7 ： else if 包头 = BB00 and 包尾 = B00B ：DeviceInforAnalysis()←EquipmentStatusInformation Protocol 8： if 队列 2 被占用（lock）：等待， else：入队列 2（设备信息队列） 9：线程池分派线程读取队列 1（数据信息队列）和队列 2（设备信息队列），重复 6-8 步且同样加锁 10：存储数据入数据库，读取队列 3（指令队列） 12：if 队列 3！= null：ControlPackage()←ControlPackage Protocol 13：如果任务少线程多则结束部分空闲线程；如果所有队列为空：仍有线程正在执行则等待，否则 pool_destroy()
Output	Data Information && Device Information && Command

在本平台所设计的算法中，Thread_pool 结构体为线程管理器，它主要作为用户交互与线程池的交互接口，结构体包含了工作线程，任务队列等引用。Thread_worker 结构体中封装了任务接口，通过自定义 DataInforAnalysis()、DeviceInforAnalysis()以及 ControlPackage()实现该接口功能，进行数据的解析和指令封装。

线程池通过调用 poolinit()函数在线程池中创建一些的线程，线程的数目可以根据线程池中的实际情况进行修改。通过 PoolAddWorker()函数将上位机中的数据任务加入到工作队列中，该函数中会判断线程数目是否达到线程最大数目。根据结果决定该任务是否可以得到运行如果任务队列中的任务都执行完后，工作线程在执行任务时调用 ThreadtaskCall()函数，在该函数中会将任务队列中的任务放到工作线程中执行。

线程的数量会根据队列中的任务数来动态的增加或消除线程池中的线程。如果任务的数量过高，就会创建新的工作线程来对任务进行处理。当任务变少时会销毁一些工作线程，来尽量降低系统的开销。当线程池中的所有的工作线程都在处理任务时，包括原有的工作线程和新创建的线程都处于工作状态，并且线程数已达到上限，队列中的任务就只能选择等待。在等到工作线程处理完之后等待的任务才有机会被处理。

当任务队列中所有任务都被处理完，工作线程的状态就会变为阻塞状态。同时工作线程需要等待上位机出现新的任务，工作线程再次被唤起进行工作。如果想要销毁线程池，需要调用 pooldestroy()方法进行处理。该方法在销毁线程池之前会对工作线程的状态进行查看判断。如果线程池有任务正在进行，则会等待线程池中任务执行完后再进行销毁。

3.4 高并发数据接口性能测试

为了对高并发数据接口在大数据量，实时性方面的表现，本小节通过三方面对其接口进行测试：大数据量处理性能测试、最高处理性能时的线程数以及单条数据等待时间。

（1） 大数据量处理性能测试 为了检测高并发数据接口在大数据量时的处理性能，分别对单线程、多线程以及线程池技术进行大数据量处理测试。实验采用 Mercury Load-Runner 7.6 工具用来模拟用户发送数据。其中本实验采用的工具是专门用来进行对应用进行测试的。首先需要更改一下该模拟工具中的的虚拟用户数，分别选取进行实验的值分别为 1、2、3、4、5，同时指定每个用户每次发出 1000 条数据。根据对实验结果进行分析可以看出，线程池的数据处理性能在单线程、多线程以及线程池中相比较而言是最好的，多线程稍次之，单线程相比之下性能最差。线程性能测试实验结果如图 3-3 所示。

了本次实验。本次实验是分别测试线程池和单线程对一条数据处理的等待时间，并将两种线程的等待时间的结果进行对比。实验进行一百次，使用一百条数据进行测试，并记录其等待时间。单条数据等待时间的测试结果如图 3-5 所示。通过实验结果图可以看出单线程处理一条数据的平均时间大概为 1.23ms，而线程池处理一条数据的平均时间大概为 1.14ms。并且从图中可以看出单线程处理时，数据的等待时间变化波动较大。而通过线程池处理数据时，时间波动比较小，等待时间较均衡。因此，通过本次实验可以得出结论，线程池技术在处理单条数据时的性能要优于单线程技术。同时线程池技术不但可以降低等待时间，还能使等待时间较为平衡，可以防止数据长时间得不到处理而影响工作效率。

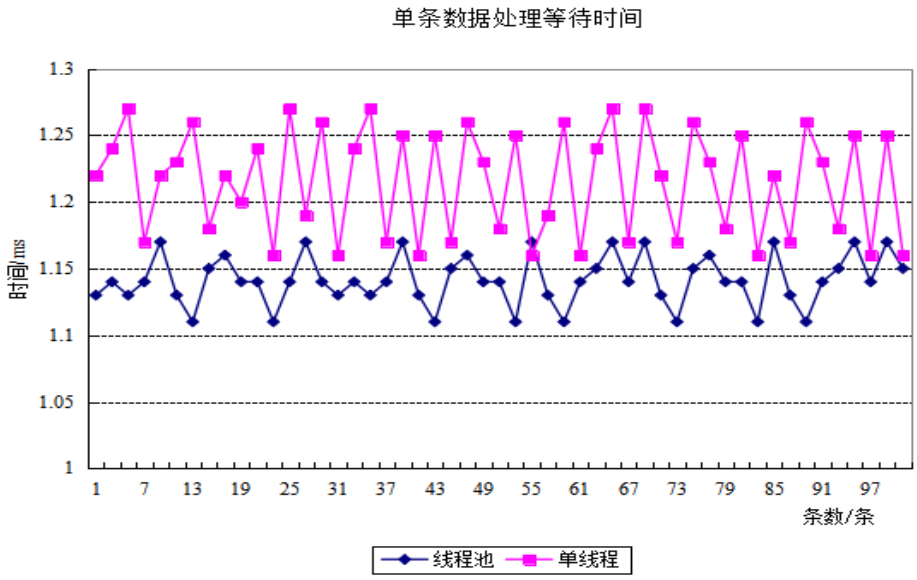

图 3-5　单条数据处理等待时间测试

综上所述，通过使用高并发接口在较大数据量上处理时单条数据平均等待时间为 1.14ms。结合延迟测试中，上位机在接收数据、处理、转发，最终到达服务器耗时 2100ms，通过分析得出网络状态是影响数据传送耗时的主要原因，所以可通过提高通信网络状况，来提升数据实时性。

3.5　本章小结

本章主要对高并发数据接口的设计做了详细的介绍。

第一节介绍了所使用的相关技术，包括高并发技术和线程池技术，高并发技术主要介绍了一些性能指标和线程池的工作流程，除此之外还对单线程和线程池进行了对比和介绍。第二节比较详细的介绍了高并发数据接口的设计方案，其中具体包含接口设计流程，任务队列以及线程池管理和服务端高并发数据接收程序算法。第三节介绍了高并发数据接口的性能，主要是对本章提出的高并发数据接口的设计方

案进行了验证。

第 4 章　基于读写分离的数据库存储方案设计与优化

本文所设计的面向平台的高并发数据接口与数据库之间存在大量的数据通信。因此，平台就需要使用高性能的数据库存储方案，提高平台的高并发实时性。

4.1　需求分析

数据库是计算机用来储存数据、文字、图片等信息的“仓库” [30]。在断电的状态下不丢失数据，是软件系统中的必不可少的一部分[31]。而对于本文来说，数据库的性能方面是整个存储系统的关键环节。高并发的数据访问量平台必须要拥有一套，保证系统性能极佳的数据库存储设计方案作为支撑。但是，普通的平台在数据存储的高并发和实时性方面仍存在些许不足，主要有以下两种表现形式：

1）当较大数据量的用户在线访问平台，向服务器发送请求时，平台资源利用率会随请求数量的上升而增长。而当并发数量达到一定数目时，平台存储系统的并发处理能力会急剧降低，甚至有可能会出现服务器崩溃等现象。因此，存储系统的并发性能提高就显得尤为重要。

2）微环境监测平台在某一个时刻接收到海量实时数据时，传统的数据库储存、显示速度相对较为缓慢，虽然也可以实现数据的存储操作，但是却不能达到实际意义上的实时性的性能标准。

因此，本文旨在设计一套，基于读写分离的数据库存储优化方案，满足平台对高并发以及实时性的需求，并提升平台的并发数据存储能力，解决上述微环境监测平台存在的两点问题。

4.2　数据库存储方案及优化设计

4.2.1　基于读写分离的数据库存储方案设计

读写分离通常情况下都是针对数据库来进行设计和使用的[32]。简单来说就是，将数据库分为主数据库和从数据库，由主数据库提供数据的写入服务，在此基础上再增加一个或者多个从数据库，主要进行数据的读取服务[33]。一般来说，读写分离技术可以应用到各种不同的业务场景中，通常将凡事可以通过提供读服务，减轻主数据库读操作的压力的方案，都列为读写分离方案。目前读写分离技术已经广泛的应用在在国内外许多企业中[34]。

增加读库的存储方案主要需要解决的问题包括数据的同步、复制以及一致性等问题[35]。林怀忠等主要研究了数据的评价准则以及异步复制等相关方面。王延青等提出了“拉推式”的方案，主要解决了分布式数据库数据在数据复制时传输量大和效

率低等问题。

如何在降低网络通信成本的基础上提高存储数据的一致性和利用率成为了近几年中数据缓存存储方案的热点问题。谢骋超等是通过对哈希表进行深入研究，设计出一套基于分布式的缓存系统。黄世能等则提出了虚缓存节点的新概念，并使用不同的方案解决缓存中的一致性问题。

（1） 基于增加读库的读写分离技术 增加读库的目的是减轻主数据库的压力[36]。因此要增加从库要保证不能改变主数据库提供主要服务，在此原则上，增加一个或是多个从库实现只读的功能要求[37]。增加读库的目的主要是要解决主库的数据传输到从库的过程中数据的复制问题。通常情况下，一般的数据库本身都包含数据的复制功能[38]。然而在真正的业务生产中，需要考虑的问题很有很多，其中主要的问题有数据复制的过程中转换、延时以及过滤条件的支持等[39]。增加读库方案的系统架构图如图 4-1 所示。

微环境监测平台使用的数据是由传感器提供的环境数据，而 MongoDB 数据库作为功能最丰富的非关系型数据库，并且根据其存储特点非常适合对传感数据进行存储。因此本文选择使用 MongoDB 数据库作为读写分离的主库，为平台提供稳定的写操作服务；另一方面，MySQL 数据库是当下最流行的关系型数据库之一，同时存储性能很稳定。因此本文选用多个 MySQL 数据库作为从库，为平台提供数据的读取操作服务。

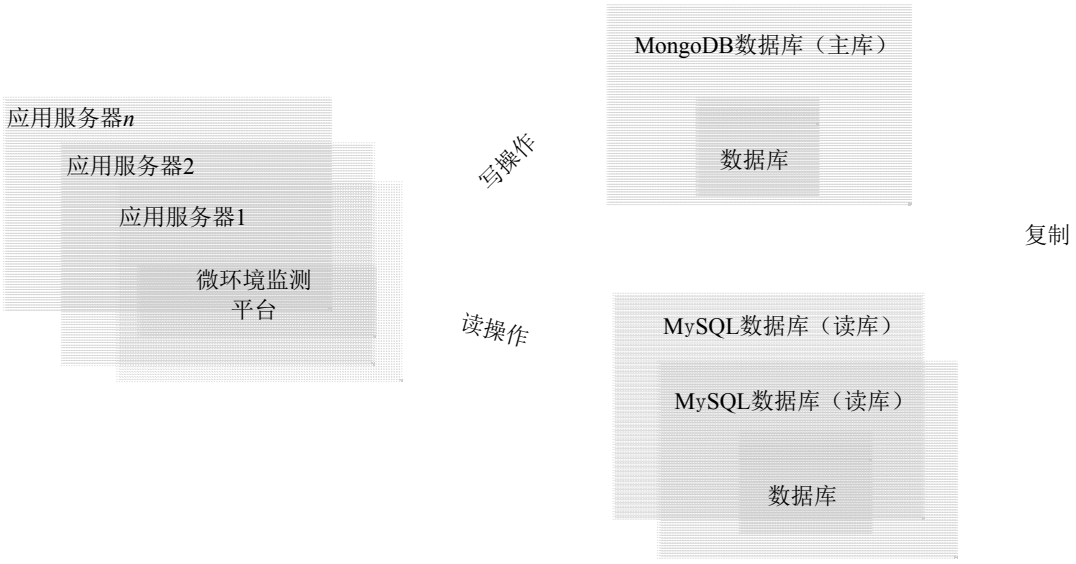

图 4-1 增加读库方案架构示意图

一般来说，数据库本身所支持的数据的复制功能通常都是有限的[40]。当平台同时进行大量的更新操作时，主数据库中数据复制时就会出现很大程度的延迟现象。数据出现延迟必然会导致在一定的时间内，用户在读库中读到的数据并不是实时数据，与实际上主库中的数据不一致的现象。此时读库中的数据就不是最新的数据，