级 1501013

号 0077

西安电子科技大学

本科毕业设计论文

题　　目　基于 Node.js 的 Web 网管系统服务器设计和实现

学　　院　通信工程学院

专　　业　通信工程

学生姓名　李星晨

导师姓名　张岗山

摘要

随着近几年 Web 技术的不断发展，基于 Web 技术的各种互联网 APP 也越来越多地出现在我们的生活中，Web 的图形化、与平台无关、分布式、动态及交互性也为网管系统提供了新的发展方向。同时 Node.js 对 IO 密集型服务器的良好支持也为 Web 网管系统提供了性能上的保障。本文主要针对 Node.js 服务器和 Web 技术特点进行分析，在两种技术融合的基础上设计一款基于 Web 的网络管理系统服务器。在设计在这款服务器的过程中遇到的最大的困难是如何及时正确的推送实时数据。

本文首先对网络管理系统进行分析，提出网管系统数据的获取修改方式和存储展示方式。其次对 HTTP 协议与 WebSocket 协议进行研究，结合 AJAX 和 WebSocket 设计网管系统服务器与 Web 客户端的通信模型。最后根据 Node.js 服务器和 MySQL 数据库设计网管系统数据模型。

依据前三章的理论将 Node.js 服务器和 Web 技术进行结合，对 Web 网管系统服务器进行功能模块的设计和实现。包括控制器模块、数据库模块、数据库访问模块和推送模块的设计和实现、数据库各表的设计和实现。

最后对 Web 网管系统服务器和 Web 客户端进行了两轮测试，分析了测试结果，并对服务器提出了优化方向，对整个 Web 网管系统 SNMP 相关部分提出了设计思路。

关键词： Node.js Web 网络管理系统 服务器

ABSTRACT

With the continuous development of Web technology in recent years, various kinds of Internet APPs based on Web technology are more and more appearing in our life. The graphical, platform-independent, distributed, dynamic and interactive Web also provides a new direction for the development of network management system. At the same time, the good support of Node.js for IO-intensive servers also provides performance guarantee for Web network management system. In this paper, the characteristics of Node.js server and Web technology are analyzed, and a Web-based network management system server is designed based on the integration of the two technologies. The biggest difficulty in designing this server is how to properly push real-time data in time.

Firstly, the network management system is analyzed, and then the methods of data acquisition and modification, the methods of data storage and display are put forward. Secondly, the HTTP protocol and WebSocket protocol are studied, and the communication model between network management system server and Web client is designed by combining AJAX and WebSocket. Finally, the data model of network management system is designed according to Node.js server and MySQL database.

According to the theory of the first three chapters, Node.js server and Web technology are combined to design and implement the functional modules of Web network management system server. Including the design and implementation of controller module, database module, database access module and push module, the design and implementation of database tables.

Finally, two rounds of tests are carried out on the server and client of the Web network management system, the test results are analyzed, the optimization direction of the server is proposed, and the design ideas of the parts about SNMP of the whole system are put forward.

Keywords: Node.js Web Network Management System Serve

目录

第一章 绪论

1.1 选题背景及意义

由于网络技术的快速发展，管理各种网络设备的任务变得越来越困难。为了确保可靠的数据传输服务，传统上使用专用管理控制台来监视和分析网络段上的流量状态。在一个经常为企业范围网络采用各种网络技术的大型组织中，传统的网络管理解决方案变得过于昂贵。有必要选择适当的网络管理标准来有效地管理设备。

简单网络管理协议(SNMP)是目前的网络管理标准。由于其架构简单，几乎每个网络设备都支持 SNMP。 嵌入在网络设备中的代理程序收集流量统计信息并将其记录在管理信息库中，网络管理员可以通过实时轮询代理来获取信息[1]。

然而，最近 Web 技术的发展为网络管理技术发展提供了新的方向。与传统网络管理平台相比，基于 Web 的网络管理有三个优点：首先，配置或监控设备不需要专门的管理软件。要利用基于 Web 的网络管理，只需在客户端运行 Web 浏览器即可。其次，通过使用基于 Web 的网络管理，代理和管理器不需要同时更新。通过基于 Web 的代理提供管理信息，应用程序可以使用 Web 浏览器的常见熟悉界面，并且浏览器适用于所有平台。下一个主要优点是应用程序的平台和位置独立性。网络管理员可以从任何平台上的任何位置访问 Web 控制的网络。所需要的只是一个通用的 Web 浏览器，它是绝大多数平台的标准软件。

1.2 国内外现状分析

集中式网络管理模式是目前使用最为普遍的一种模式。此模式的系统中需要设置专门的网络管理节点，所有被管终端被网络管理节点监视，并在网络管理节点的监视下协同工作，以此来实现网络管理功能。在集中式网络管理系统中，所有的管理软件和功能集中在专门的网络管理节点上，网络管理节点和被管终端是主从关系。

网络管理节点与被管终端通过网络通信信道相连。网络管理节点可以直接控制干预所有被管终端的配置参数，监控整个网络系统中所有设备和链路的实时运

行状态，统计整个网络系统的信息流量情况，对所有设备和链路进行故障测试、诊断和修复处理，还能远程控制被管终端。

从集中式网络管理模式的自身特点可以看出，集中式网络管理模式的优点是管理集中，网络管理员在一个位置就可以查看到所有的网络报警和事件[2]，这有助于发现故障以及确定问题的关联性。但是，管理信息集中汇总到网络管理站节点上，会导致网络管理信息流比较拥挤，管理不够灵活，管理站节点如果发生故障则有可能影响整个网络管理系统的正常工作[3]，只适合于小型局域网络、部门专用网络、统一经营的公共服务网、企业互联网络等。

1.3 论文工作内容

本论文根据目前网络管理系统的发展趋势，设计并实现一套基于 Node 的分布式 web 网管系统的服务器。这个系统以 SNMP 协议和 HTTP 协议为核心，通过 SNMP 协议实现子网信息的采集和处理，通过 HTTP 协议实现对网管数据的展示。总体来说，本片论文研究内容为 HTTP 协议部分实现对网管数据的展示，主要包括以下几个方面：

1）分析国内外网管系统的现状与发展前景以及 Web 网管系统带来的效率提升。

2）在学习 web 开发的基础上，提出 HTTP 与 SNMP 结合实现网络管理的思路。

3）对网络管理系统需要解决的问题及需求进行分析， 明确 Web 网管系统需要的功能。

4）根据基于 Web 的网络管理模型，设计一种 Web 网管系统的服务器框架，包括数据模型（网络拓扑结构、设备状态）和通信模型（动态实时推送）。

5）基于 Node.js 平台实现一种 Web 网管系统服务器端的网管功能，包括：获取网络拓扑、监视设备运行状态和向客户端推送信息。

6）与 Web 客户端相配合，实现网络拓扑和设备状态的动态更新和展示等功能。

1.4 论文章节安排

该论文根据研究内容，分为六个部分，分别为：

第一章，绪论。介绍该选题的背景与意义，并根据国内外现状进行分析，再介绍论文需要完成的任务与论文的整体规划。

第二章，Web 服务技术概述。介绍实现 web 网管系统服务器开发所涉及的主要技术，包括 HTTP 协议、WebSocket 协议、AJAX 技术、Node.js 开发环境和 MySQL 数据库。

第三章，基于 Web 的网管系统服务器总体设计。对 Web 网管系统在技术上、经济上和操作上的问题进行分析，并对产品的需求与功能进行分析，设计出 Web 网管系统的整体框架。

第四章，基于 Web 的网管系统服务器详细设计与实现。在系统整体框架构建完成后，对服务器部分各模块进行详细的设计并实现。

第五章，基于 Web 的网管系统测试与分析。在 Web 网管系统服务器与前端页面完成后，对其功能进行测试，确保其能够正确的完成在整个系统里需要完成的任务。

第六章，结束语。在所有开发测试工作完成后，对服务器部分进行总结，并提出进一步优化的思路。

第二章 Web 服务技术概述

2.1 HTTP 协议

超文本传输协议(HTTP)是一种基于 TCP/IP 通信协议的用于从万维网(WWW)服务器传输超文本到本地浏览器的传输协议。下面对 HTTP/HTTPS 进行简述。

2.1.1 HTTP 概述[4]

HTTP 是一个无状态、无连接的应用层协议，具有简捷、灵活的特点，常用 80 端口。

HTTPS 在 HTTP 下加入 SSL 层，用于加密敏感信息，使用 443 端口，简单讲是安全版 HTTP。

它工作于客户端-服务器架构上：浏览器作为 HTTP 客户端通过统一资源定位符(URL)向 HTTP 服务器即 WEB 服务器发送请求，WEB 服务器根据获取的请求向客户端即浏览器发送响应数据。

HTTP 的工作步骤如下：

1.客户端与 Web 服务器建立 TCP 连接

2.发送 HTTP 请求

3.服务器接受请求并返回 HTTP 响应

4.释放连接 TCP 连接

5.客户端浏览器解析响应内容

2.1.2 HTTP 请求[5]

HTTP 请求是客户端向服务端发送请求动作，告知要求。

HTTP 报文格式如下：

HTTP 请求包括状态行、请求头、空行、请求主体四部分，如下表所示：

表 2.1 HTTP 请求报文

请求方法	空格	URL	空格	协议版本	回车符	换行符	状态行
头部字段名	冒号	值		回车符	换行符		请求头
…							

1）状态行：包括请求方式 Method、资源路径 URL、协议版本 Version；

- 请求方式：

HTTP1.0 定义了三种请求方法： GET, POST 和 HEAD 方法。

HTTP1.1 新增了五种请求方法：OPTIONS, PUT, DELETE, TRACE 和 CONNECT 方法。

其中最常用的是 GET 和 POST。

- GET：请求指定的页面信息，并返回实体主体。
- HEAD：类似于 get 请求，只不过返回的响应中没有具体的内容，用于获取报头
- POST：向指定资源提交数据进行处理请求，数据被包含在请求体中。
- PUT：从客户端向服务器传送的数据取代指定的文档的内容。
- DELETE：请求服务器删除指定的页面。
- CONNECT：HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务器。
- OPTIONS：允许客户端查看服务器的性能。
- TRAC：回显服务器收到的请求，主要用于测试或诊断。

GET 和 POST 请求的区别：

GET 提交的数据会附在 URL 之后、即 URL 参数部分，数据大小受 URL 长度限制。

POST 提交的数据放置在是 HTTP 包的包体中，数据大小不受限制，安全性比 GET 高。

- URL：

统一资源定位符(URL)是一种从互联网上获取资源位置和访问方法的简洁表示，包含两个部分：

- 模式：它告诉浏览器如何处理文件，最常用的模式是 HTTP。

- 地址：包含域名、端口、虚拟目录、文件名、参数、锚。

例如：http://www.example.com:80/public/index.jsp?key:value#anchor

- 模式：http
- 域名：www.example.com
- 端口(非必要)：80
- 虚拟目录(非必要)：/public /
- 文件名(非必要)：index.jsp
- 参数(非必要)：key:value
- 锚(非必要)：anchor

2）请求头：包括字符集、消息正文长度、主机与端口和 Cookie 等信息；

3）请求正文：HTTP 请求的数据。

2.1.3 HTTP 响应[6]

HTTP 响应是服务端根据客户端发送的请求，做出具体动作，把结果回应给客户端。

HTTP 请求包括状态行、响应头、空行、响应主体四部分，如下表所示：

表 2.2 HTTP 响应报文

<table>
<tr><td>协议版本</td><td>空格</td><td>状态码</td><td>空格</td><td>状态码描述</td><td>回车符</td><td>换行符</td><td>状态行</td></tr>
<tr><td>头部字段名</td><td>冒号</td><td colspan="2">值</td><td>回车符</td><td>换行符</td><td rowspan="3">响应头</td></tr>
<tr><td colspan="5">…</td></tr>
<tr><td>头部字段名</td><td>冒号</td><td colspan="2">值</td><td>回车符</td><td>换行符</td></tr>
<tr><td>回车符</td><td>换行符</td><td colspan="5" rowspan="2"></td></tr>
<tr><td colspan="3">响应数据</td><td>响应主体</td></tr>
</table>

状态行：包括协议版本 Version、状态码 Status、状态码描述；

状态码

- 1xx：请求已接受，继续处理请求。
- 2xx：请求处理完成。
- 3xx：请求重定向
- 4xx：客户端错误

- 5xx：服务器错误

常见状态码如下表所示：

表 2.3 常见状态码表

状态码	状态信息
200	请求成功
301	请求永久重定向
302	请求临时重定向
304	请求重定向到客户端本地缓存
400	请求存在语法错误
401	请求存在语法错误
404	请求未授权
500	资源不存在

2.2 AJAX 技术

AJAX(异步 JavaScript 和 XML)是一种交互式网页应用的网页开发技术。

2.2.1 AJAX 概述

AJAX 并非一种新的技术，它是在不重新加载网页的情况下，更新局部网页的解决方法。

AJAX 是以下多种技术的结合[7]：

- 使用 CSS 和 XHTML 表示
- 使用 DOM 交互显示
- 使用 XMLHttpRequest 和服务器进行异步通信
- 使用 js 绑定和调用

XMLHttpRequest 对象[8]

XMLHttpRequest 是 ajax 的核心机制，是一种支持异步请求的技术。它的属性有：

- onreadystatechange 每次状态改变所触发事件的事件处理程序

- responseText 返回数据的字符串形式
- responseXML 返回数据的 DOM 文档形式
- status 服务器返回的状态码
- statusText 伴随状态码的字符串信息
- readyState 对象状态值

  0 (未初始化)

  1 (初始化)

  2(发送数据)

  3(数据传送中)

  4(完成)

2.2.2 AJAX 缺陷[9]

1）破坏了浏览器后退功能

由于浏览器只能记下历史记录中的静态页面，在动态更新页面后，点击返回按钮，用户返回到前一个页面而非前一个状态。

2）网络延迟

动态更新页面时，由于用户发出请求到服务器发出响应之间存在时间间隔，XMLHttpRequest 未完成时，页面状态等待更新，可能导致用户感到厌烦。

2.3 WebSocket 技术

WebSocket 是一种在单个 TCP 连接上进行全双工通信的协议。

2.3.1 WebSocket 概述

相对于 HTTP 协议来说，WebSocket 是一个持久化的协议，客户端和服务端只需要完成一次握手，就可以创建持久性连接，并进行双向数据传输。

2.3.2 WebSocket 优点[10]

1）较少的控制开销

在连接创建后，客户端和服务端之间交换数据时，用于协议控制的数据包头部相对较小。在不包含扩展的情况下，对于服务端到客户端的内容，此头部大小只有 2 至 10 字节（和数据包长度有关）；对于客户端到服务端的内容，此头部还

需要加上额外的 4 字节的掩码。相对于 HTTP 请求每次都要携带完整的头部，此项开销显著减少了。

2）更强的实时性

由于协议是全双工的，所以服务端可以随时主动给客户端下发数据。相对于 HTTP 请求需要等待客户端发起请求服务端才能响应，延迟明显更少。

3）更好的二进制支持

WebSocket 定义了二进制帧，相对 HTTP，可以更轻松地处理二进制内容。

2.4 Node.js 服务器

随着 web 的发展以及 Web 开发者的不断增加，JavaScript 变得越来越流行，Node.js 的推出使得 Web 开发者可以更加方便快速的搭建服务器程序。

Node.js 是一个基于 Chrome JavaScript 运行时建立的一个平台。它采用事件驱动 I/O 服务端 JavaScript 环境，基于 Google 的 V8 引擎，以速度非常快，性能非常好的特点成为 Web 服务器技术的新宠儿。

Node.js 与传统后端语言相比具有以下特点[11]：

1）单线程

在很多服务器端语言中，会为每一个客户端创建新的线程，这使得相同容量内存的服务器可同时支持用户变得少，因此想要连接更多用户就需增加服务器的数量，这样一来硬件成本自然就提高很多。然而当 Node.js 执行程序时，并不为所有客户的连接创建新的线程，而仅使用一个线程，先执行好前面一个路径，然后才能执行后面一个路径。采用单线程的好处是在程序执行时，完全不用考虑线程安全，不必担心出现死锁问题，一个进程中只保持动态请求处理线程，因为每个并发请求都会占用一定的内存，所以使用 Node.js 的这一特点使得内存利用变得更高效。

2）异步非阻塞 I/O

异步与同步是相对而言的，主要区别在于进行数据访问的时候，对应用程序的调用是否能立即返回。有部分语言将程序设计为同步 I/O 的模型，后续任务都需要等待 I/O 的完成。在等待过程中，CPU 不能得到充分的利用。为了使 CPU 得到充分利用，采用异步 I/O 方式，这样做节省时间，省去各个调用之间的等待时间，操作一结束，可以立刻通过回调进行函数处理，这样一来，极大地

提高了程序的执行效率。在处理异步 I/O 的同时，线程中必须存在事件循环，不断检查程序中是否有未处理完成的事件，并依此处理，在非阻塞模式下，单线程只能处理单个任务，此时 CPU 的利用率最高。

3）事件驱动

Node.js 本质是一个框架，而且是基于事件的框架，Node.js 凭借这一特点在众多后端技术之中脱颖而出。Node.js 使用事件驱动模型，事件驱动模型的优点是可扩展性高且高效，在网页服务器接收到请求时，Node.js 利用事件驱动模型，将其关闭然后进行处理, 处理完成后再去处理下一个 Web 请求。可以将此过程看作是触发事件, 先关闭这个事件驱动, 然后对其进行处理, 目的是防止二次触发。在事件驱动模型中，利用一个主循环来监听整个事件，当检测到事件时就执行回调函数，Node 使用事件机制解决所有的异步操作。

2.5 MySQL 数据库

数据库(Database)是管理信息系统的核心。数据库是以一定组织方式储存在一起的，能为多个用户共享的，具有尽可能小的冗余度的、与应用彼此独立的相互关联的数据集合[12]。

数据库产生于上世纪五十年代，随着信息技术和市场的发展，特别是上世纪 90 年代后面，数据库的管理主要的工作不在是对数据的存储和处理，此时则转变为对数据管理的方式进行管理，数据库有很多种类型，从最简单的存储有各种数据的表格到能够进行海量数据存储的大型数据库系统都在各个方面得到了广泛的应用。

在日常的工作与活动之间，数据库可以把和日常工作相关的数据放进一个库中，根据系统的需要进行处理，如很多企业将员工的基本信息如名字、年龄、性别能放到一个数据库表格中，很多这样的表进行重复则构成了数据库[13]。

数据库具有如下主要优点：1.实现数据共享。用户可以通过许多方式通过提供的接口对数据库中的数据进行访问；2.减少数据冗余；3.数据的独立性；4.数据实现集中控制；5.数据一致性和可维护性，以确保数据的安全性和可靠性；6.故障恢复[14]。

2.5.1 MySQL 概述[15]

MySQL 由瑞典 MySQL AB 公司开发，目前属于 Oracle 旗下产品。MySQL 是最流行的关系型数据库管理系统之一，在 WEB 应用方面，MySQL 是最好的 RDBMS 应用软件。

MySQL 是一种关系数据库管理系统，关系数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。

MySQL 所使用的 SQL 语言是用于访问数据库的最常用标准化语言。MySQL 软件采用了双授权政策，分为社区版和商业版，由于其体积小、速度快、总体拥有成本低，尤其是开放源码这一特点，一般中小型网站的开发都选择 MySQL 作为网站数据库。

由于其社区版的性能卓越，搭配 PHP 和 Apache 可组成良好的开发环境。

2.5.2 MySQL 特性

MySQL 使用 C 和 C++编写，并使用了多种编译器进行测试，保证了源代码的可移植性。它支持 AIX、FreeBSD、HP-UX、Linux、Mac OS、NovellNetware、OpenBSD、OS/2 Wrap、Solaris、Windows 等多种操作系统，并且为多种编程语言提供了 API。这些编程语言包括 C、C++、Python、Java、Perl、PHP、Eiffel、Ruby,.NET 和 Tcl 等。MySQL 支持多线程，可以充分利用 CPU 资源；优化的 SQL 查询算法，能有效地提高查询速度。它既能够作为一个单独的应用程序应用在客户端服务器网络环境中，也能够作为一个库而嵌入到其他的软件中。MySQL 还提供多语言支持，常见的编码如中文的 GB 2312、BIG5，日文的 Shift_JIS 等都可以用作数据表名和数据列名。

MySQL 提供 TCP/IP、ODBC 和 JDBC 等多种数据库连接途径，提供用于管理、检查、优化数据库操作的管理工具，支持大型的数据库，可以处理拥有上千万条记录的大型数据库，支持多种存储引擎。，MySQL 是开源的，所以你不需要支付额外的费用，它使用标准的 SQL 数据语言形式，对 PHP 有很好的支持。MySQL 可以定制，它采用了 GPL 协议，可以通过修改源码来开发自己的 MySQL 系统。

第三章 基于 Web 的网管系统服务器总体设计

3.1 通用性问题分析

3.1.1 技术可行性

根据课题对系统功能、操作平台及操作便利性等各项约束条件，从技术角度来看，web 网络管理系统的实现是可行的。本系统使用 Node.js 语言，以 AJAX 和 WebSocket 为通信方式，MySQL 为数据库。Node.js 对 IO 密集型服务器的良好支持、HTML5 与 socket 网络编程技术的日渐成熟为本系统的实现提供坚实基础。

3.1.2 经济可行性

相对于传统的集中式网络管理系统而言，web 系统具有开发周期短、开发成本低、维护成本低、平台独立性好等特点。本系统的实现在经济上是可行的，开发 web 软件与服务器不需要太多的开发成本，并且由于界面与逻辑的分离，产品迭代升级不需要推倒重来，可以节约许多费用。本系统的开发支出包括：硬件设备：PC 机，软件设备：WINDOWS/LINUX 操作系统、Node 开发库、MySQL 数据库，其他支出：软件开发费用、软件维护费用。

3.1.3 操作可行性

由于地理隔离及网络设备种类繁杂，人们对网络管理系统的要求也逐渐提高，这时，需要一种全新的工具可以在任意地点管理所有的网络设备来提高网络管理效率，web 网管系统的出现解决了这一问题。由于互联网的全球性，网络管理者可以在任意地点通过互联网访问 web 网管系统来管理不同地域的网络设备。不同等级的管理者可以获取不同的管理权限，这也使网络管理更加系统规范。由此可见，web 网络管理系统具有良好的操作可行性。

3.2 系统需求分析

根据基于 Web 的网络管理模型，设计一种 Web 网管系统的服务器框 架，包括：数据模型（网络拓扑结构、设备状态）和通信模型（动态实时 推送）；

基于 Node.js 平台实现一种 Web 网管系统服务器端的网管功能，包括：获取网络拓扑、监视设备运行状态和向客户端推送信息；

本系统是基于 Web 的网络管理系统，系统分为两个部分：用户管理部分和设备数据更新部分。

3.2.1 用户管理部分

用户管理部分包括登录模块、设备管理模块、告警推送模块。

- 登录模块,该模块负责完成登录、退出操作，并且赋予或取消管理权限。
- 设备管理模块,该模块负责获取设备信息、获取链路信息、获取告警信息。
- 告警推送模块,该模块负责推送用户在线时新增或删除的告警信息。

3.2.2 设备数据更新部分

设备数据更新部分包括登录模块、修改信息模块。

- 登录模块,该模块负责完成登录、退出操作，并且赋予或取消高级管理权限。
- 修改信息模块,该模块负责完成设备信息改变时发送的修改数据库与推送指令。

3.3 系统总体框架设计

网络管理系统需要完成信息采集、信息处理、信息存储、信息展示四个任务。本人采取分布式的数据采集模式完成了整体框架模型设计,如下图所示：

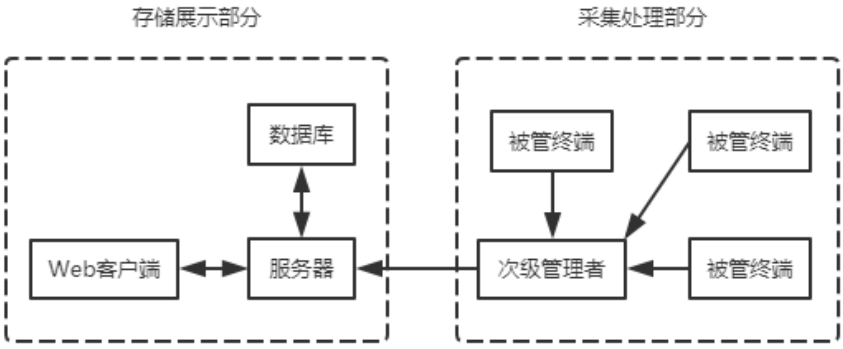

图 3.1 系统总体框架图

整个框架模型分为两个部分，分别为采集处理部分和存储展示部分。

1) 采集处理部分包括被管终端和次级管理者。

被管终端即被管理的网络设备，负责采集本机的实时 CPU、内存、磁盘利用

率、路由表等设备信息，并将其发送给次级管理者。

次级管理者即子网中的数据采集点，负责将接收的信息统一整理，并通知存储展示部分更新数据。

2) 存储展示部分包括服务器、数据库和 Web 客户端。

服务器负责根据采集处理部分发送的通知修改数据库对应数据，以及为 Web 客户端提供信息服务。

数据库存储整个网管系统的所有数据。

Web 客户端负责从服务器获取数据并将其转化为图形展示给管理者。

该系统的获取数据流程如下。

图 3.2 获取数据流程图

管理者通过 Web 客户端，向 Web 服务器发送获取数据请求，Web 服务器获取请求并解析，从数据库获取对应数据，并返回响应给 Web 客户端，Web 客户

端解析数据并展示给管理者。

该系统的更新数据流程如下。

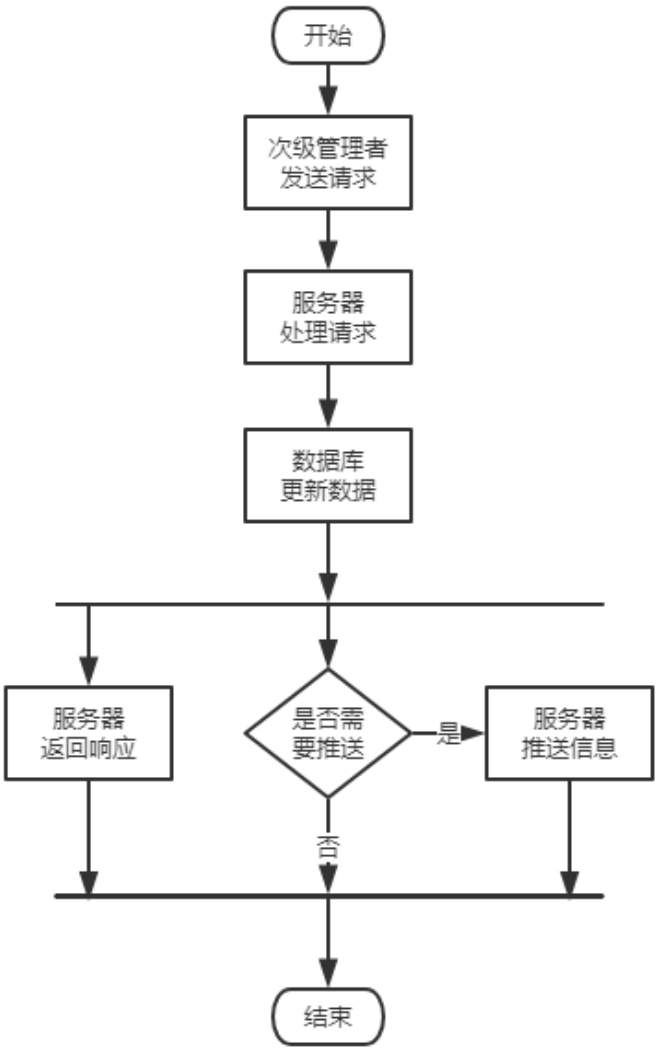

图 3.2 更新数据流程图

被管终端状态改变，次级管理者获取改变的信息并进行整理，发送给服务器，服务器根据获取的数据修改数据库，并且判断是否需要推送，如需推送则向 Web 客户端推送该信息。

在本课题中，本人主要负责存储展示部分中的服务器和数据库模块的开发。

3.4 服务器关键技术

服务器主要使用 Node.js（v 10.15.1）开发。

3.4.1 AJAX

服务器使用 express（v 4.16.4）框架实现 AJAX 接口，为 Web 客户端与次级管理者提供获取和修改数据的接口，其中使用了 express-session（v 1.15.6）、express-mysql-session（v 2.1.0）、body-parser（v 1.18.3）中间件。

- express-session 为服务器提供 session，用于记录用户登录状态与权限。
- express-mysql-session 用于将 session 数据存储到数据库中。
- body-parser 用于解析请求主体。

3.4.2 WebSocket

服务器使用 ws（v 6.2.1）库实现 WebSocket，为 Web 客户端提供推送功能。

3.4.3 MySQL

服务器使用 mysql（v 2.16.0）库实现与 MySQL 数据库的连接，并实现数据库的增、删、改、查。

数据库使用 MySQL（v 8.0.15）社区版。

3.5 Web 服务器与网管服务器接口模型

3.5.1 网管服务器接口

本系统网管服务器接口基于 HTTP 协议，调用接口前需要先登录，用于获取高级权限。

接口数据采用 JSON 的数据格式，HTTP 请求方式（Method）为 POST，请求数据格式如下：

```
{
    "id":[ ],
    "data":{ }
}
```

其中 id 和 data 至少存在一个。

响应数据格式如下：

```
{
    "code": 200 || 300,
    "data": [ ],
```

```
    "message": "success"
}
```

其中 code 为响应状态码，200 为成功、300 为失败；message 为伴随响应状态码的字符串信息；data 为响应数据主体。

3.5.2 Web 服务器接口

本系统 Web 服务器接口基于 HTTP 协议和 WebSocket 协议，分别用于获取网管信息和获取推送。其中获取网管信息需要先登录，用于获取普通权限，获取推送需要 WebSocket 长连接获取授权。

1）获取网管信息接口采用 JSON 的数据格式，HTTP 请求方式（Method）为 POST，请求数据格式如下：

```
{
    "id":[ ],
    "data":[ ]
}
```

其中 id 和 data 有且只有一个。

响应数据格式如下：

```
{
    "code": 200 || 300,
    "data": [ ],
    "message": "success"
}
```

其中 code 为响应状态码，200 为成功、300 为失败；message 为伴随响应状态码的字符串信息；data 为响应数据主体。

2）获取推送接口采用 JSON 的数据格式，推送格式如下：

```
{
    "message": "inform",
    "data": {
        "name": "link" || "machine" || "topology" || "warning",
        "result": linkDetail || machineDetail || undefined || {"type":"add" || "delete",
```

```
"data": warningDetail, "id":[ ]}
      }
}
```

其中 message 为推送标题；data.name 为推送类型；data.result 为推送的信息主体。linkDetail 为链路信息对象，machineDetail 为设备信息对象，warningDetail 为告警信息对象。

3.5.3 通用接口

通用接口基于 HTTP 协议，包括登录接口和退出接口。数据格式为 x-www-form-urlencoded，请求方式为 POST。登录接口请求数据中字段 account 为账号，password 字段为密码，退出接口无请求数据；响应数据皆与网管服务器接口响应数据格式相同。

第四章 基于 Web 的网管系统服务器详细设计与实现

4.1 服务器整体结构的设计和实现

为提高 Web 客户端性能，防止信息更新尤其是拓扑图更新时的页面闪动，本系统采用前后端分离的开发方法，视图与模型在前端，控制器在后端，最后将前端打包文件放在/public 文件夹内并通过 express 框架的 express.static('public')实现静态页面挂载。

服务器包含数据库模块，数据库访问模块、控制器模块、和推送模块、静态页面。

项目目录如下：

```
---controller
---dao
---database
---public
---websocket
---index.js
```

数据库模块路径为"/database"，负责数据库的连接与 sql 命令的执行。数据库访问模块路径为"/dao"，负责生成 sql 命令。控制器模块路径为"/controller"，负责提供 http 接口。推送模块路径为"/websocket"，负责推送信息。

/index.js 部分代码段如下：

```
const events=require('events');
global.EventEmitter=new events.EventEmitter();
const database=require('./database/main');
const wsToken=require('./database/wsToken');
const controller=require('./controller/main');
const webSocket=require('./websocket/main');
```

其中 global.EventEmitter 创建 EventEmitter 实例，为 WebSocket 推送提供全局事件触发器。

控制器模块由/index.js 调用，数据库访问模块由控制器模块调用，数据库模

块绑定在全局并由数据库访问模块调用，推送模块绑定在全局并通过事件触发。

服务器整体结构如下：

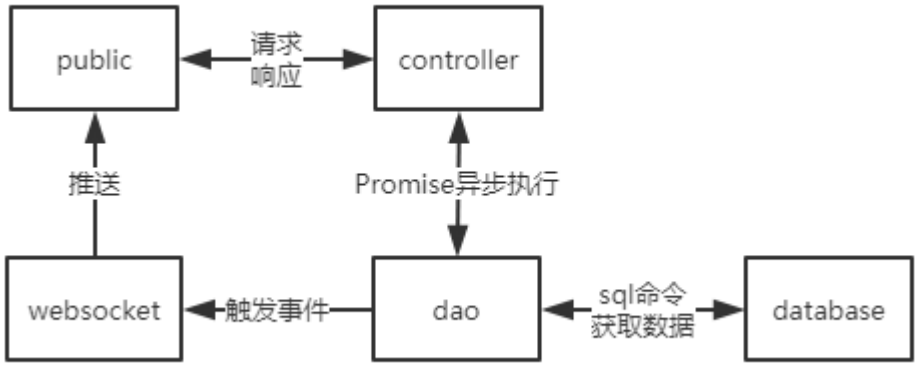

图 4.1 服务器整体结构图

控制器模块获取 HTTP 请求，根据请求的行为要求依次调用数据库访问模块中的具体方法，数据库访问模块生成 sql 命令，调用数据库模块并执行，将返回的数据传给控制器模块，控制器模块回复响应并决定是否调用推送模块推送信息。

4.2 数据库模块的设计和实现

数据库模块路径为"/database"，负责数据库的连接与 sql 命令的执行。

数据库模块目录如下：

```
---database
------dbInfo.js
------main.js
------wsToken.js
```

/database/dbInfo.js 用于保存数据库连接信息，其中 xidiannms 用于存储网管系统数据，xidiannms_session 存储 session 信息。

/database/dbInfo.js 主要代码段如下：

```
module.exports={
   xidiannms:{
      host     : 'localhost',
      user     : 'root',
      password : '1121807045',
```

```
    database : 'xidiannms',
    multipleStatements: true
  },
  xidiannms_session:{
    host     : 'localhost',
    user     : 'root',
    password : '1121807045',
    database : 'xidiannms_session',
  }
}
```

/database/main.js 通过 mysql 第三方库完成服务器与数据库的连接，并且定义全局方法 dbQuery()实现执行 sql 命令，用于执行网管系统数据的增、删、改、查。/database/wsToken.js 用于 WebSocket 权限查询。

4.3 数据库访问模块的设计和实现

数据库访问模块路径为"/dao"，负责生成 sql 命令。

数据库访问模块目录如下：

```
---dao
------SQLString
---------link.js
---------machine.js
---------other.js
---------warning.js
------link.js
------machine.js
------other.js
------warning.js
```

/dao/*分别定义了链路、设备、告警与其他四类 Promise 异步方法，多个方法可使用 Promise.then、Promise.all 和 Promise.race 拼接完成更加复杂的指令。

4.3.1 关于链路操作的所有方法

/dao/link.js 中定义了以下方法：

- get()：查询链路

通过链路 ID 查询链路，返回链路信息。

- getByMachineId()：查询链路(ByMachineId)

通过设备 ID 查询与此设备连接的链路，返回链路 ID。

- add()：增加链路

增加链路，链路两端设备必须存在，返回新增链路 ID。

- deleteById()：删除链路(ById)

通过链路 ID 删除链路，返回被除链路 ID。

- deleteByMachine()：删除链路(ByMachine)

通过设备 ID 删除与之相连的链路。（预留）

- deleteByLink()：删除链路(ByLink)

通过链路信息删除链路，返回被删链路 ID。

- changeDetail()：修改链路详情

修改链路详情，返回被修改链路 ID。

- changeStatus()：修改告警标识

通过告警信息修改链路告警标识警信息。

4.3.2 关于设备操作的所有方法

/dao/machine.js 中定义了以下方法：

- get()：查询设备

通过设备 ID 查询设备，返回设备信息。

- add()：增加设备

增加设备，返回新增设备 ID。

- deleteById()：删除设备(ById)

通过设备 ID 删除设备，返回被删设备 ID。（预留）

- deleteByMachine()：删除设备(ByMachine)

通过设备信息删除设备，返回被删设备 ID。

- changeDetail()：修改设备详情

修改设备详情，返回被设备链路 ID。

- changeStatus()：修改告警标识

通过告警信息修改设备告警标识警信息。

4.3.3 关于告警操作的所有方法

/dao/warning.js 中定义了以下方法：

- get()：查询告警

通过告警 ID 查询告警，返回告警信息。

- add()：增加告警

增加告警，返回新增告警 ID。

- deleteById()：删除告警(ById)

通过告警 ID 删除设备，返回被删告警 ID。（预留）

- deleteByWarning()：删除告警(ByWarning)

通过告警信息删除告警，返回被删告警 ID。

- deleteByLinkId()：删除告警(ByLinkId)

通过链路 ID 删除告警，返回被删告警 ID。

- deleteByMachineId()：删除告警(ByMachineId)

通过设备 ID 删除告警，返回被删告警 ID。

4.3.4 其他异步方法

/dao/other.js 中定义了以下方法：

- httpRes()：HTTP 响应
- wsSend_inform()：WebSocket 推送
- signIn()：登录
- signOut()：退出
- getAuthority()：判断权限（普通权限）
- setAuthority()：判断权限（高级权限）
- getStaticData()：获取静态信息

4.3.5 SQL 语句片段

/dao/SQLString/*中统一保存所有的 SQL 语句片段与拼接方法。

4.4 控制器模块的设计和实现

控制器模块路径为"/controller"，负责提供 http 接口。

控制器模块目录如下：

```
---controller
------ctl
---------ctl_addLink.js
---------ctl_addMachine.js
---------ctl_addWarning.js
---------ctl_changeLinkDetail.js
---------ctl_changeMachineDetail.js
---------ctl_deleteLink.js
---------ctl_deleteMachine.js
---------ctl_deleteWarning.js
---------ctl_getLink.js
---------ctl_getMachine.js
---------ctl_getWarning.js
---------ctl_signIn.js
---------ctl_signOut.js
---------ctl_staticData.js
------middleware
---------filter.js
---------route.js
------main.js
```

4.4.1 控制器中间件

控制器模块使用了 body-parser、express-session、express-mysql-session 第三方中间件与多个自定义中间件，在获取 HTTP 请求后保存 session 用于记录登录状态，用/controller/middleware/filter.js 过滤所有的错误地址，用 CORS 代码段

实现跨域访问，再通过 body-parser 解析 JSON 数据，最后将数据传入 /controller/middleware/route.js。

CORS 代码段

```
app.use(function(req, res, next){
    res.setHeader('Access-Control-Allow-Origin', 'http://localhost:8080');
    res.setHeader('Access-Control-Allow-Credentials', true);
  res.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, PATCH, DELETE');
  res.setHeader('Access-Control-Allow-Headers', 'X-Requested-With,content-type');
  next();
})
```

4.4.2 控制器接口

/controller/middleware/route.js 提供了所有 HTTP 接口，如下：

- 登录：接口为'/signIn'，请求方式为'POST'，不需要权限。
- 退出：接口为'/signOut'，请求方式为'POST'，不需要权限。
- 获取静态表：接口为'/staticData'，请求方式为'POST'，不需要权限。
- 查询设备：接口为'/getMachine'，请求方式为'POST'，需要普通权限。
- 查询链路：接口为'/getLink'，请求方式为'POST'，需要普通权限。
- 查询告警：接口为'/getWarning'，请求方式为'POST'，需要普通权限。
- 删除设备：接口为'/deleteMachine'，请求方式为'POST'，需要高级权限。
- 删除链路：接口为'/deleteLink'，请求方式为'POST'，需要高级权限。
- 删除告警：接口为'/deleteWarning'，请求方式为'POST'，需要高级权限。
- 增加设备：接口为'/addMachine'，请求方式为'POST'，需要高级权限。
- 增加链路：接口为'/addLink'，请求方式为'POST'，需要高级权限。
- 增加告警：接口为'/addWarning'，请求方式为'POST'，需要高级权限。
- 修改设备详情：接口为'/changeMachineDetail'，请求方式为'POST'，需要高级权限。

- 修改链路详情：接口为’/changeLinkDetail'，请求方式为'POST'，需要高级权限。

4.4.3 控制器行为流程

/controller/ctl/*包含所有请求处理行为，处理行为为异步流程（即 Promise 对象，前方法的返回值作为参数传给后一个方法）。

所有行为流程如下：

- getWarning

```
查询告警 > HTTP响应
```

- addWarning

```
增加告警 > 查询告警 > 修改链路/设备状态 > WS推送
         > HTTP响应
```

- deleteWarning

```
查询告警 > 修改链路/设备状态 > 删除告警(byWarning) > WS推送
                                                  > HTTP响应
```

- getLink

```
查询链路 > HTTP响应
```

- addLink

```
增加链路 > WS推送
         > HTTP响应
```

- deleteLink

```
查询链路 > 删除链路(byLink) > 删除告警(ByLinkId) > WS推送
                            > HTTP响应
```

- changeLinkDetail

```
修改链路详情 > 查询链路 > WS推送
             > HTTP响应
```

- getMachine

```
查询设备 > HTTP响应
```

- addMachine

```
增加设备 > WS推送
         > HTTP响应
```

- deleteMachine

```
查询设备 > 删除设备(ByMachine) > 删除告警(ByMachineId)
                                 查询链路(ByMachineId) > 删除链路(ByLink) > 删除告警(ByLinkId) > WS推送
                               > HTTP响应
```

图 4.2 控制器行为流程

- ctl_getMachine.js：查询设备

先通过设备 ID 查询设备，再调用 HTTP 响应。

- ctl_getLink.js：查询链路

先通过链路 ID 查询链路，再调用 HTTP 响应。

- ctl_getWarning.js：查询告警

先通过告警 ID 查询告警，再调用 HTTP 响应。

- ctl_deleteMachine.js：删除设备

先查询设备，在根据设备信息删除设备，再同时执行通过设备 ID 删除告警信息、通过设备 ID 查找链路、HTTP 响应，查询链路完成后通过链路信息删除链路，再通过链路 ID 删除告警，最后 WebSocket 推送拓扑变化。

- ctl_deleteLink.js：删除链路

先通过链路 ID 查询链路，再通过链路信息删除链路，再同时执行通过链路 ID 删除告警信息、HTTP 响应，最后在删除告警信息成功后通过 WebSocket 推送拓扑变化。

- ctl_deleteWarning.js：删除告警

先通过告警 ID 查询告警信息，再通过告警信息修改链路或设备状态，再通过告警信息删除告警，最后同时执行 HTTP 响应和 WebSocket 推送告警信息更新。

- ctl_addMachine.js：增加设备

先添加设备，再同时执行 HTTP 响应和 WebSocket 推送拓扑改变。

- ctl_addLink.js：增加链路

先添加链路，再同时执行 HTTP 响应和 WebSocket 推送拓扑改变。

- ctl_addWarning.js：增加告警

先添加设备，再同时执行 HTTP 响应和通过告警 ID 查询告警信息，再通过告警信息修改链路或设备状态，最后 WebSocket 推送告警信息更新。

- ctl_changeMachineDetail.js：修改设备详情

先执行修改设备详情，再同时执行通过设备 ID 查询设备信息和 HTTP 响应，查询设备完成后通过 WebSocket 推送设备信息。

- ctl_changeLinkDetail.js：修改链路详情

先执行修改链路详情，再同时执行通过链路 ID 查询链路信息和 HTTP 响应，查询链路完成后通过 WebSocket 推送链路信息。

4.5 推送模块的设计和实现

推送模块路径为"/websocket"，负责推送信息。

推送模块目录如下：

```
---websocket
------main.js
------message.js
------send.js
```

WebSocket 连接成功后服务器会向 Web 客户端推送 sec-websocket-key 字段，此字段是 Web 客户端 WebSocket 连接的唯一序列 KEY。Web 客户端登录后将获取的 token 字段通过 WebSocket 发送给服务器即可获取推送权限。

/websocket/message.js 为 WebSocket 提供验证 token 并并获取权限的方法。/websocket/send.js 包括所有推送服务，并根据次级管理者的 HTTP 请求的接口推送不同服务：当接口为'/addWarning'或'/deleteWarning'时推送告警改变，当接口为'/changeMachineDetail'时推送改变的设备信息，当接口为'/changeLinkDetail'时推送改变的链路信息，当接口为'/addLink'、'/addMachine'、'/deleteLink'或'/deleteMachine'时推送拓扑改变。

4.6 数据库的设计和实现

本系统总共创建了两个数据库，分别为 xidiannms 和 xidiannms_session，其中 xidiannms_session 存储服务器 session 信息，xidiannms 存储网络管理系统信息。以下主要说明 xidiannms 数据库的具体字段。

xidiannms 数据库总共包括十张数据表，分别为 account（账户信息表）、warning（告警信息表）、warning_level（告警级别表）、warning_type（告警类型表）、create_way（产生方式表）、machine_type（设备类型表）、link_type（链路类型表）、status（状态表）、machine（设备信息表）、link（链路信息表），可分为 4 类：

4.6.1 账户信息

账户信息存储在 account（账户信息表）里：

表 4.1 账户信息表设计表

键	类型	备注
account_id	int(11)	账号 ID（主键）
account	varchar(255)	账户名称

password	varchar(255)	密码
email	varchar(255)	邮箱
authority	int(11)	权限值
dataChange_createTime	datetime(3)	创建时间
dataChange_changeTime	datetime(3)	最后修改时间

4.6.2 告警信息

告警信息存储在 warning（告警信息表）、warning_level（告警级别表）、warning_type（告警类型表）、create_way（产生方式表）里，其中 warning 为所有告警信息，其他为静态表，补充描述 warning 中部分键值：

表 4.2 告警信息表设计表

键	类型	备注
warning_id	int(11)	告警 ID（主键）
warning_level	int(11)	告警级别
warning_type	int(11)	告警类型
create_way	int(11)	告警产生方式
warning_aim	varchar(255)	告警目标类型
warning_aim_id	int(11)	告警目标 ID
warning_time	int(11)	告警时间
dataChange_createTime	datetime(3)	创建时间
dataChange_changeTime	datetime(3)	最后修改时间

表 4.3 告警等级表设计表

键	类型	备注
warning_level_id	int(11)	告警等级 ID（主键）
warning_level	int(11)	告警级别
description	varchar(255)	告警级别描述
dataChange_createTime	datetime(3)	创建时间
dataChange_changeTime	datetime(3)	最后修改时间

表 4.4 告警类型表设计表

键	类型	备注
warning_type_id	int(11)	告警类型 ID（主键）
warning_type	int(11)	告警类型
description	varchar(255)	告警类型描述
dataChange_createTime	datetime(3)	创建时间
dataChange_changeTime	datetime(3)	最后修改时间

表 4.5 告警产生方式表设计表

键	类型	备注
create_way_id	int(11)	告警产生方式 ID（主键）
create_way	int(11)	告警产生方式
description	varchar(255)	告警产生方式描述
dataChange_createTime	datetime(3)	创建时间
dataChange_changeTime	datetime(3)	最后修改时间

4.6.3 设备信息

设备信息存储在 machine（设备信息表）、machine_type（设备类型表）、status（状态表）里，其中 machine 为所有设备信息，其他为静态表，补充描述 machine 中部分键值：

表 4.6 设备信息表设计表

键	类型	备注
machine_id	int(11)	设备 ID（主键）
machine_type	int(11)	设备类型
name	varchar(255)	设备名称
machine_status	int(11)	设备状态
ip_address	varchar(255)	IP 地址
SNMP_address	varchar(255)	SNMP 地址

memory_used_ratio	double	内存使用率
memory_total	double	内存总量
cpu_used_ratio	double	CPU 使用率
cpu_total	double	CPU 总量
product_business	varchar(255)	产商
port_total	int(11)	端口总数
physical_port_total	int(11)	物理端口总数
description	varchar(255)	描述
start_time	datetime(3)	开始运行时间
dataChange_createTime	datetime(3)	创建时间
dataChange_changeTime	datetime(3)	最后修改时间

表 4.5 设备类型表设计表

键	类型	备注
machine_type_id	int(11)	设备类型 ID（主键）
machine_type	int(11)	设备类型
description	varchar(255)	设备类型描述
dataChange_createTime	datetime(3)	创建时间
dataChange_changeTime	datetime(3)	最后修改时间

表 4.5 状态表设计表

键	类型	备注
status_id	int(11)	状态类型 ID（主键）
status	int(11)	状态
description	varchar(255)	状态描述
dataChange_createTime	datetime(3)	创建时间
dataChange_changeTime	datetime(3)	最后修改时间

4.6.4 链路信息

设备信息存储在 link（链路信息表）、link_type（链路类型表）、status（状态表）里，其中 link 为所有链路信息，其他为静态表，补充描述 link 中部分键值：

表 4.6 设备信息表设计表

键	类型	备注
link_id	int(11)	链路 ID（主键）
link_type	int(11)	链路类型
link_status	int(11)	链路状态
from_machine	int(11)	源设备
from_machine_port	varchar(255)	源设备端口
to_machine	int(11)	目的设备
to_machine_port	varchar(255)	目的设备端口
brand_width	double	带宽
speed	double	速率
in_speed	double	入速率
out_speed	double	出速率
brand_width_used_ratio	double	带宽利用率
in_brand_width_used_ratio	double	入带宽利用率
out_brand_width_used_ratio	double	出带宽利用率
dataChange_createTime	datetime(3)	创建时间
dataChange_changeTime	datetime(3)	最后修改时间

表 4.5 链路类型表设计表

键	类型	备注
link_type_id	int(11)	链路类型 ID（主键）
link_type	int(11)	链路类型
description	varchar(255)	链路类型描述
dataChange_createTime	datetime(3)	创建时间
dataChange_changeTime	datetime(3)	最后修改时间

第五章 基于 Web 的网管系统测试与分析

前面各章节完成了对基于 Node.js 的 web 网管系统服务器的设计和实现，本章对该系统进行功能测试，以下为测试的环境搭建和详细测试过程。

5.1 运行环境的搭建

由于服务器与客户端是分别开发的，开始测试时客户端并未完成，所以进行了接口测试和前后端联合测试两次功能测试。

5.1.1 接口测试环境

第一次网管系统服务器的接口测试使用的是本地测试，测试环境如下：

表 5.1 接口测试环境

处理器	Intel(R) Core(TM) i5-4200H
内存	4GB
硬盘	1TB
操作系统	Windows 10 家庭中文版
运行环境	Node.js v 10.15.1
数据库	MySQL 社区版 v 8.0.15
测试软件	Postman、Chrome 浏览器

5.1.2 前后端测试环境

第二次测试为服务器与客户端联合测试，使用阿里云服务器，测试环境如下：

表 5.2 前后端联合测试环境

处理器	1 核
内存	2GB
硬盘	40GB
操作系统	Windows Server 2016 数据中心版 64 位中文版
运行环境	Node.js v 10.15.1
数据库	MySQL 社区版 v 8.0.15

测试软件	Postman、Chrome 浏览器

5.2 网络管理系统测试分析

5.2.1 接口测试

第一次接口测试，由于没有客户端，使用 Postman 发送 HTTP 请求，并且使用 Chrome 浏览器控制台执行如下代码段模拟 WebSocket 连接接受推送信息。

```
(function(){
    var ws=new WebSocket('ws://localhost:3000');
    ws.onopen=function(){
        console.log(`WebSocket('ws://localhost:3000')`);
    }
    ws.onmessage=function(event){
        console.log(JSON.parse(event.data));
    }
    window.wsSend=function(data){
        ws.send(JSON.stringify(data))
    };
})()
```

测试结果如下：

表 5.3 接口测试结果

测试接口	请求方式	响应是否正确	推送类型	测试结果是否正常
/signIn	POST	正确	无	正常
/staticData	POST	正确	无	正常
/getWarning	POST	正确	无	正常
/deleteWarning	POST	正确	warning	正常
/addwarning	POST	正确	warning	正常
/getLink	POST	正确	无	正常

/deleteLink	POST	正确	topology	正常
/addLink	POST	正确	topology	正常
/changeLinkDetail	POST	正确	link	正常
/getMachine	POST	正确	无	正常
/deleteMachine	POST	正确	topology	正常
/addMachine	POST	正确	topology	正常
changeMachineDetail	POST	正确	machine	正常
/signOut	POST	正确	无	正常

测试结果分析：所有的接口都可以正常运行，修改网管系统数据后，WebSocket 也可以及时正确的推送信息。这说明本系统的服务器与数据库部分已经可以完成在整个系统中需要执行的任务。

5.1.1 前后端测试

在前端页面开发完成后，对本系统进行了第二次测试，第二次测试为客户端与服务器联合测试，具体测试结果如下：

表 5.4 前后端联合测试结果

功能模块	测试点	测试步骤	实测试结果
登录	登录接口、WebSocket 授权	打开登录页面，输入正确的账号密码	登录成功，WebSocket 授权成功
拓扑图	获取设备信息、获取链路信息、获取告警信息	1、点击 Topology 按钮进入拓扑图页面。 2、打开 Postman，通过 addWarning、addMachine、addLink、deleteWarning、deleteMachine、deleteLink 接口修改对应信息。 3、打开 Postman，通过 changeLinkDetail、changeMachineDetail 接口修改对	拓扑图正确生成，告警信息正确加载，信息修改后拓扑图正确更新、告警信息正确更新

		应信息。	
静态信息表	获取静态信息表	分别点击 Static 标签下的 MachineType、LinkType、Status、WarningType、WarningLevel、CreateWay，查看其信息	静态表加载正确
重新连接	Session 记住登录状态、WebSocket 重新连接	1、断开互联网，点击 Topology 按钮进入拓扑图页面，使用 Postman 通过对应接口修改数据库数据。 2、连接互联网，点击 Topology 按钮进入拓扑图页面，使用 Postman 通过对应接口修改数据库数据。	断网后无法获取实时更新，重新连接后信息自动更新，并可以继续接收推送

测试结果分析：经过多次测试，客户端均可以正确地获取数据库信息并及时的获取且处理推送，这表明本系统可以简单实现网络管理功能。

第六章 结束语

本文通过对 SNMP 协议与 HTTP 协议进行分析，结合当前市场对网络管理的新需求和 Web 技术的发展情况，设计了一种基于 Node.js 的 Web 网络管理系统，并实现了其中的服务器与数据库部分。服务器以 Web 为基础，以 B/S 模式实现数据的交换。

6.1 论文工作总结

纵观全文，本论主要通过对 Web 网络管理系统进行需求分析、重点技术分析、总体系统设计设计了一套分布式 web 网管系统，并对基于 Node.js 的 Web 网络管理系统服务器进行功能模块设计实现与多次测试。具体包括如下几个方面：

1）本文从用户交互性与体验角度出发，提出了以 AJAX 和 WebSocket 这两种种异步传输技术来对数据报文进行传输的方案。利用 AJAX 机制将数据报文包装在 HTTP 请求主体里，然后通过 HTTP 协议进行传输或利用 WebSocket 将推送报文以 JSON 形式主动推送。

2）设计实现了基于 Promise 的异步链式请求处理机制。将所有请求动作拆分成多个可复用的 Promise 具体方法，通过不同顺序组合具体方法实现请求的处理。

3）设计实现了 WebSocket 事件，通过不同的事件传递不同种类的命令或数据。

4）完成了数据库设计。参考其他网络管理系统的网管信息数据，设计了一套完整的网管信息数据库，包括了用户数据，设备、链路数据和告警数据。

5）分别在本地和云服务器对服务器进行了两轮测试，并完成了测试分析，结果符合预期要求。

6.2 后续工作展望

由于时间等条件的限制，本文所作的工作还存在许多需要改进和完善的地方，主要包括以下几个方面：

1）本文只包括了 Web 网管系统服务器与数据库的设计和实现，除去相关毕设的客户端部分，还有被管终端和次级管理者的子系统需要实现。

2）目前服务器只实现了监控方面的接口，其他远程控制与管理方面的接口还需继续完善。

3）目前服务器只在功能上证明了基于 Node.js 的 Web 网管系统的可行性，并没有在性能上提出太多要求，如果要投入使用，还有许多地方有待优化。

[1] 参考文献

[1] Ching-Wun Tsai,Ruay Shiung Chang. SNMP through WWW. International Journal of Network Management. 1998

[2] 刘洁，基于 SNMP 协议的分布式网络管理系统设计与实现，西安电子科技大学硕士论文，2010

[3] 陈明，局域网络教程，清华大学出版社，2004

[4] https://blog.csdn.net/sinat_39480731/article/details/82288915

[5] https://www.cnblogs.com/jiu0821/p/5641600.html

[6] https://www.jianshu.com/p/0015277c6575

[7] https://www.cnblogs.com/kennyliu/p/3876729.html

[8] https://www.cnblogs.com/zsber/p/9526108.html

[9] 梁宏亮，面向物联网应用和终端的开放式管理平台设计与实现，北京邮电大学硕士论文，2012

[10] https://blog.csdn.net/libaineu2004/article/details/82774638

[11] https://blog.csdn.net/dancheng1/article/details/77803147

[12] 胡雅颖，基于物化视图的查询系统研究与实现，河北工业大学硕士论文，2006

[13] 李怀磊，基于 Web 的教务管理系统的设计与实现，西安电子科技大学硕士论文，2013

[14] 马东旭，基于.NET 的绩效管理系统的设计与实现，西安电子科技大学硕士论文，2014

[15] http://www.ijava.com/javajiaocheng/mysql.html

致谢

四年的大学生活即将结束，在这四年的成长道路中，在西电这个大家庭里，我已经从一个懵懂的新生成长成即将踏入社会的毕业生。对此，我要感谢身边每一个帮助过我的人。

首先感谢我的导师张岗山教师。本文是在张岗山教授的悉心指导下完成的。在将近半年的时间里，从论文的立题到完成，他花费了许多时间和经历对我进行指导，并提出了宝贵的意见。

其次感谢我的项目合作者罗财生。在服务器的实现阶段，他给了我很大的帮助，包括数据库的设计和实现、代码的规范化和测试。

还要感谢我的舍友，他们在设计阶段给了我很多有意义的建议和启发。

最后，特别感谢我的父母、家人，他们默默地给予我极大的支持和理解。