1. **绪论**
   1. 选题背景及意义

随着网络技术的高速发展，计算机网络的应用的深度和广度不断提高，日渐深入我们的工作和生活，并发挥着越来越重要的作用。广阔的应用空间对网络运行的稳定性、健壮性以及安全性提出了很高的要求。为保证网络高速稳定地运行，需要网络管理人员能够方便地监控网络运行的状态并及时地发现问题以便加以解决。由于地理上的隔离、网络设备种类的繁杂等原因，网络管理需要有新的工具使网络管理员可以管理处于不同地理位置的网络设备，并且对网络管理行为加以简化，提高网络管理的效率。

近年来，随着Web技术的不断发展和广泛应用，Web与网络管理相结合成为网络管理技术发展的一个重要方向。与传统网络管理平台相比，基于Web的网络管理具有简单易用、节省费用、独立于平台以及具有分布性和开放性的优点，故而在网络管理中成为更好的选择。

* 1. 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，其在网络系统中设置专门的网络管理节点，所有的网管代理在网络管理站的监视和控制下协同工作，实现集成的网络管理。管理软件和管理功能主要集中在网络管理站上，网络管理站节点与网管代理节点是主从关系。

网络管理站通过网络通信信道与所有网管代理相连。网络管理站可以对所有代理的配置参数进行直接控制和干预，可以实时监视全网代理的运行状态，统计和掌握全网的信息流量情况，可以对全网进行故障测试、诊断和修复处理，还可以对被管代理进行远程加载、转储以及远程启动等控制。网管代理定时向网络管理站提供自己位置信息和必要的管理信息。

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，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，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. 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. 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。

1. **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

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

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 | | |
| … | | | | | | | | | |
| 头部字段名 | | 冒号 | 值 | | 回车符 | | | 换行符 | |
| 回车符 | 换行符 | |  | | | | | | | | | |
| 请求内容 | | | | | | | 请求主体 | | | | | |

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

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

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 协议版本 | | 空格 | 状态码 | 空格 | | 状态码描述 | | | 回车符 | | 换行符 | **状态行** |
| 头部字段名 | | 冒号 | 值 | | 回车符 | | | 换行符 | | **响应头** | | |
| … | | | | | | | | | |
| 头部字段名 | | 冒号 | 值 | | 回车符 | | | 换行符 | |
| 回车符 | 换行符 | |  | | | | | | | | | |
| 响应数据 | | | | | | | **响应主体** | | | | | |

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：

* 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

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

* 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。

* 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

* 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。

* 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。

* 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：

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。

1. 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。

1. 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

MySQL是一个关系型数据库管理系统，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。

2.5.1 MySQL概述

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。

1. **基于Web的网管系统服务器总体设计**
   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。

* 1. 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。
  1. 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：

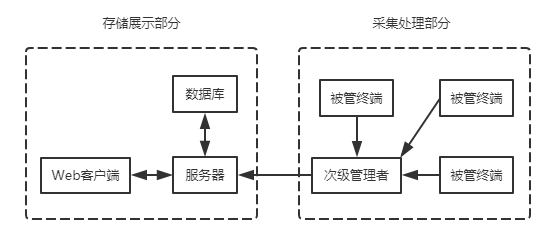


图3.1 系统总体框架图

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。

* 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。

* 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。

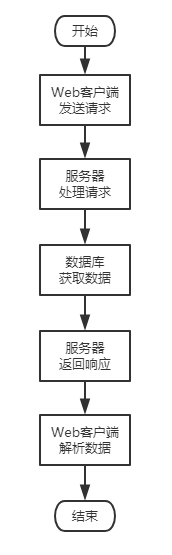


图3.2 获取数据流程图

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。

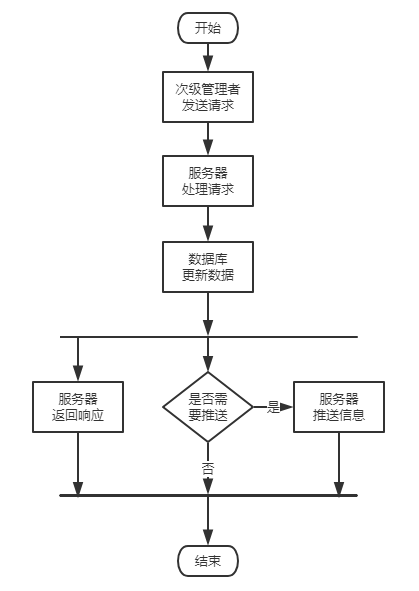


图3.2 更新数据流程图

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。

* 1. 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。

* 1. Web服务器与网管服务器接口模型

1. **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：

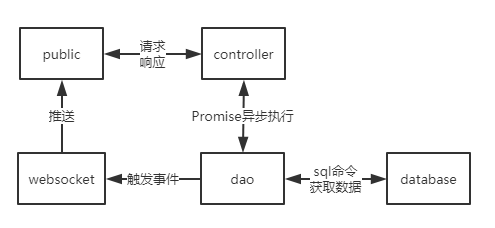


图4.1 服务器整体结构图

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：



图4.1 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数据库总共包括九张数据表，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）。
* account：账户信息表包括七个字段，分别是account\_id（账号ID）、account（账户名称）、password（密码）、email（邮箱）、authority（权限值）、dataChange\_createTime（创建时间）、dataChange\_changeTime（最后修改时间），其中account\_id为主键。
* Warning：告警信息表包括九个字段，分别是warning\_id（告警ID）、warning\_level（告警级别）、warning\_type（告警类型）、create\_way（产生方式）、warning\_aim（告警目标类型）、warning\_aim\_id（告警目标ID）、warning\_time（告警时间）、dataChange\_createTime（创建时间）、dataChange\_changeTime（最后修改时间），其中warning\_id为主键。
* warning\_level：告警级别表为静态表，总共包含五个字段，分别是warning\_level\_id（告警级别ID），warning\_level（告警级别），description（告警级别描述），dataChange\_createTime（创建时间），dataChange\_changeTime（最后修改时间），其中warning\_level\_id为主键。
* warning\_type：告警类型表为静态表，总共包含五个字段，分别是warning\_type\_id（告警类型ID），warning\_type（告警类型），description（告警类型描述），dataChange\_createTime（创建时间），dataChange\_changeTime（最后修改时间），其中warning\_type\_id为主键。
* create\_way：产生方式表为静态表，总共包含五个字段，分别是create\_way\_id（产生方式ID），create\_way（产生方式），description（产生方式描述），dataChange\_createTime（创建时间），dataChange\_changeTime（最后修改时间），其中create\_way\_id为主键。
* machine\_type：设备类型表为静态表，总共包含五个字段，分别是machine\_type\_id（设备类型ID），machine\_type（设备类型），description（设备类型描述），dataChange\_createTime（创建时间），dataChange\_changeTime（最后修改时间），其中machine\_type\_id为主键。
* link\_type：链路类型表为静态表，总共包含五个字段，分别是link\_type\_id（链路类型ID），link\_type（链路类型），description（链路类型描述），dataChange\_createTime（创建时间），dataChange\_changeTime（最后修改时间），其中link\_type\_id为主键。
* status：状态表为静态表，总共包含五个字段，分别是status\_id（状态ID），status（状态类型），description（状态描述），dataChange\_createTime（创建时间），dataChange\_changeTime（最后修改时间），其中status\_id为主键。
* machine：设备信息表包括十七个字段，分别是machine\_id（设备ID），machine\_type（设备类型），name（设备名称），machine\_status（设备状态），ip\_address（IP地址），SNMP\_address（SNMP地址），memory\_used\_ratio（内存使用率），memory\_total（内存总量），cpu\_used\_ratio（CPU使用率），cpu\_total（CPU总量），product\_business（产商），port\_total（端口总数），physical\_port\_total（物理端口总数），description（描述），start\_time（开始运行时间），dataChange\_createTime（创建时间），dataChange\_changeTime（最后修改时间），其中machine\_id为主键。
* Link：链路信息表包括十六个字段，分别是link\_id（链路ID），link\_type（链路类型），link\_status（链路状态），from\_machine（源设备），from\_machine\_port（源设备端口），to\_machine（目的设备），to\_machine\_port（目的端口），brand\_width（带宽），speed（速率），in\_speed（入速率），out\_speed（出速率），brand\_width\_used\_ratio（带宽利用率），in\_brand\_width\_used\_ratio（入带宽利用率），out\_brand\_width\_used\_ratio（出带宽利用率），dataChange\_createTime（创建时间），dataChange\_changeTime（最后修改时间），其中link\_id为主键。

1. 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 前后端联合测试环境

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

5.1.1 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 前后端联合测试

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接口修改对应信息。 | 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 |
| 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。