基于TCP/IP协议和Modbus协议的楼宇灯控系统

灯具开关管理、关联设置、实时控制

摘要

在社会发展的过程中，信息流、物流、资金流是社会发展和进步的三大要素。而在这三大要素中，由于信息流处在中间环节，即信息流的流通可带动并促进物流和资金流的流通，所以信息流是社会发展的重中之重。在机械工业时代，信息的传播由于受制于空间上的限制，时效性和有效性都处于很低的水平，这时的社会各个生产要素发展缓慢，各个行业也按照过去几十年、上百年甚至上千年的模式发展着，变化的只是效率或者操作方式，而根本的模式却并没有大的变化。但步入21世纪，伴随着军用或科研用的计算机技术和信息技术的快速普及和发展，社会进入信息化时代，各行各业发生了翻天覆地的变化。一直处在传统领域的安防和楼宇灯控也不例外。在传统楼宇灯控中，只有普通的布线方式、供电方式和管理方式，计算机参与度很少或者没有，这样的楼宇灯控系统，在过去各大企业以及大型场所使用也并无问题，因为在过去一般组织和个人是没有能力和需求使用这些楼宇灯控系统，而一旦使用了这些楼宇灯控系统的企业和场所基本都能负担的起其高昂的建设成本、电力成本和人力管理成本。近年来，由于信息技术的普及，以及摩尔定律的作用，小型处理器功能越来越强劲，体积却越来越小，这就为新兴智能楼宇灯控的产生创造了条件。由于基于计算机网络技术、电子信息技术、WEB2.0技术以及硬件智能化（物联网）技术的智能楼宇灯控系统在建设成本、使用方式、功能选择和节约电力资源等方面的优势越来越突出，行业内新增量的楼宇灯控系统几乎都会选择上述的智能楼宇灯控系统，已经存量的传统楼宇灯控系统也在陆陆续续的进行改造，以达到信息化、智能化的管理。这些大型企业和大型场所，在使用智能楼宇灯控系统后，在节省电力资源和科学化管理水平方面提升明显，这也为智能楼宇灯控系统的普及和推广创造了很好的口碑效应。

为了解决以上阐述的问题,我们本着实践出真知的态度,在此次毕设选题中选择了基于TCP/IP协议和Modbus通讯协议的楼宇智能灯控系统。本智能楼宇灯控系统后端使用SpringMVC框架完成，使用ExtJS、Ajax框架等技术实现人性化界面的设计和便捷的用户体验，采用MySQL数据库完成数据的存储。以上技术均为所学知识，所以具有技术可行性。

本文首先从智能楼宇灯控系统的背景、现状、意义等方面进行了简要说明；然后对智能楼宇灯控系统开发过程中使用到的系统开发环境、对SpringMVC框架、及软件工程中各种设计模式等技术作了简要介绍；接着详细阐述智能楼宇灯控系统设计过程中的需求分析、功能设计、数据库表设计、系统详细设计，且给出了系统实现过程中关键部分的代码实现，并阐述了本系统如何采用上述技术实现项目的开发，给出了主要功能的实现效果；文章还对开发过程中使用到的SpringMVC框架、Modbus通讯协议等新技术进行了详细的描述；文章最后对项目实施过程进行了总结与展望。

**关键字：**SpringMVC，ExtJS，TCP/IP，Modbus，智能照明

Building lighting control system based on TCP / IP protocol and Modbus protocol

-- Building lighting control system based on TCP / IP protocol and Modbus protocol

**ABSTRACT**

In the process of social development, information flow, logistics, capital flow is the three elements of social development and progress. In these three elements, because the flow of information in the middle part, that is, the flow of information can drive and promote the flow of logistics and capital flow, so the flow of information is the most important social development. In the era of mechanical industry, the dissemination of information due to constraints on the space constraints, timeliness and effectiveness are at a very low level, when the development of various factors of social development is slow, the various industries in accordance with the past few decades, The development of the model for thousands of years, the change is only the efficiency or mode of operation, and the fundamental model is not a big change. But into the 21st century, with the military or scientific research and use of computer technology and information technology rapid popularization and development of society into the information age, all walks of life has undergone enormous changes. Has been in the traditional areas of security and building lighting control is no exception. In the traditional building lighting control, only the ordinary way of wiring, power supply and management methods, computer participation is little or no, such a building lighting control system, in the past, large enterprises and large places are also no problem, because in the In the past, general organizations and individuals were unable and demanded to use these building lighting control systems, and once they used these buildings, the light control system of enterprises and places were able to afford the high cost of construction, electricity costs and manpower management costs. In recent years, due to the popularity of information technology, and the role of Moore's Law, small processors more and more powerful, the volume is getting smaller and smaller, which for the emerging intelligent building lighting control to create the conditions. As the intelligent building lighting control system based on computer network technology, electronic information technology, WEB2.0 technology and hardware intelligent (Internet of Things) technology has become more and more prominent in terms of construction cost, use mode, function selection and saving power resources , The industry will add the amount of building lighting control system will almost all of the intelligent building lighting control system, has been the stock of traditional building lighting control system is also a continuation of the transformation, in order to achieve information and intelligent management. These large enterprises and large sites, in the use of intelligent building lighting control system, in the conservation of power resources and scientific management level to enhance the obvious, which is intelligent building lighting control system to promote and promote the creation of a good reputation effect.

In order to solve the above problem, we have chosen the intelligent lighting control system based on TCP / IP protocol and Modbus communication protocol in this thesis. The intelligent building lighting control system using the SpringMVC framework to complete the use of ExtJS, Ajax framework and other technologies to achieve user-friendly interface design and user experience, the use of MySQL database to complete the data storage. The above techniques are learned, so it is technically feasible.

In this paper, the background, present situation and significance of the intelligent building lighting control system are briefly described. Then, the system development environment used in the development of the intelligent building lighting control system, the design pattern of the SpringMVC framework and the software engineering And then gives a brief introduction to the intelligent building lighting control system design process needs analysis, functional design, database table design, system detailed design, and gives the key part of the system implementation process code, and elaborated How to use the above technology to achieve the development of the project, given the main function of the implementation of the effect of the article also used in the development process of the SpringMVC framework, Modbus communication protocol and other new technologies were described in detail; the final article on the project implementation process Made a summary and outlook.

**Key Words:**SpringMVC ExtJS TCP/IP Modbus Intelligent lighting

# 绪论

## 研究背景

在信息技术的推动下，现代化楼宇使用了越来越多的具有自控功能的设备，这些设备功能各不相同，布线方式相互交错，复杂性极高。但在所有的这些系统中，有一个系统无论是大型还是小型建筑，且是否是智能化的，它都在这些自控系统中占据非常重要的位置，它就是楼宇灯控系统。据统计，传统建筑中有20% - 30%的电力是被照明系统给消耗的，这在整个电力消耗中占比是比较大的。为了解决上述传统楼宇灯控系统构建复杂，使用不够智能，且由于不够智能进而造成电力消耗巨大，造成企业运营成本高居不下不说，还造成资源的浪费的问题。我们在此次设计选题中，决定就智能楼宇灯控系统进行研究。该系统是把WEB2.0技术和AT89C51嵌入式单片机处理技术进行整合，即软硬件相结合的系统设计。该系统可根据现场情况，选择多种控制方式（单开，光感，红外等）实现对灯具状态的控制，实现楼宇照明的人性化、个性化。

传统的楼宇灯控系统一般使用普通交流电电缆，将各个节点的连接电缆都接入总设备间。这就造成了楼宇间的布线非常复杂，极其雍余，一方面这部分成本占整个楼宇灯控系统成本的很大一部分，增加了企业的负担，另一方面，由于存在灯回路的辐射干扰，空间的电磁环境也会被污染。基于AT89C51嵌入式单片机的控制器普及使用为现代化楼宇智能照明提供了一个新的途径。它采用网络控制技术，使得照明灯的电力线路不再使用普通电线而是使用网线提供电力，同时网线还用来传输控制信号，灯具的接口使用RJ45水晶头，使得布线过程大大简化，维护和升级的成本大大降低。

## 应用现状

从20世纪60年代开发了白炽灯、荧光灯、高强度放电灯所使用的电子调光器，到20世纪90年代以来，国外以计算机技术为基础开发出的灯光自动调光系统、自动关停系统和自动补偿系统，也称“智能照明”的新型照明控制系统，并已有定型产品得以良好的推广和运用，使建筑照明由传统控制走向计算机控制或无人控制的新领域。自1984年美国建成第一座智能建筑以来的十几年中，在世界范围内，智能建筑以一种崭新的面貌和技术，迅速在各地展开。尤其是亚洲的日本、新加坡等国家和地区，为了适应智能建筑的发展，进行了大量的研究和实践，相继建成了一批具有智能化的建筑。

## 实现目标和意义

### 实现目标

智能照明控制系统是将过去需要人工操作的步骤用程序集成起来，通过计算机信息技术，网络控制技术以及AT89C51小型CPU技术整合起来的高技术的信息系统。它的核心是智能，该系统的智能主要是指通过人为的给系统设置一些参数，在无人的情况下系统将通过现场实时数据进行状态的更新。该系统并为控制灯具的开关模拟了单开、红外、光感和调光等模式，实现多样化的灯具状态控制。它的另一个特点是节能，既然是照明系统，在传统的照明控制方式中，不可避免的会出现电力资源的浪费现象，但是使用此系统因为添加了智能控制，可以实现随用随开，随走随关，这些操作都是自动完成的，因此极大的方便了用户的同时也解决了资源浪费问题。

### 实现意义

进入新世纪，我国经济发展迅速，工业和信息产业已成为我国的支柱产业，但是发展的同时也产生了另一个问题，即电力不足问题，智能照明系统的节能特点对于缓解我国目前电力紧缺的问题会有很大帮助。进入2016年，我国进入十三五阶段，国家顶层对于“互联网+”的模式非常重视，对于各行各业实现信息化国家非常支持。因此开发这样一个系统，对于贯彻落实国家“互联网+”的战略布局，实现该领域的信息化是非常有意义的。

## 研究过程

由于本智能楼宇灯控系统的设计出发点是以现实需求为本，以解决日常使用痛点为目标。故整个系统功能模块较多，下面一一列举说明：权限管理、楼宇管理、系统管理、灯具管理、关联设置、实时控制这六个功能模块。其中权限管理、楼宇管理和系统管理前三个功能模块主要是对现实中的实体比如楼宇、房间等进行管理，并对其建立之间的所属关系、映射关系，本功能模块主要由本次毕设同组另一位同学负责实现。而我在此次毕业设计中主要负责该系统灯具开关管理、关联设置与实时控制的实现。在本次设计工作中，我利用在学校所学的数据库知识，网络知识、Java Socket网络编程知识以及自学的Modbus工控协议进行该部分的开发工作。通过将所学理论知识与实际编码遇到的问题相结合，既更加牢固的掌握了专业知识，又对之前错误的理解纠偏纠正，且学习到了学校里没有开设课程的知识。

# 系统开发环境及技术简介

## 2.1.USR-TCP232-200串口服务器

网络转串口模块USR-TCP232-200是用来将TCP数据交换报文或UDP数据交换报文与RS232接口数据实现透明传输的设备，模块体积小巧，功耗低，搭载ARM处理器，速度快，稳定性高。这是一款多功能型嵌入式以太网串口数据转换模块, 它内部集成了 TCP/IP 协议栈, 用户利用它可以轻松完成嵌入式设备的网络功能, 节省人力物力和开发时间, 使产品更快的投入市场, 增强竞争力。模块集成 10/100M 自适应以太网接口, 串口通信最高波特率高达 230.4Kbps, 具有TCP Server, TCP Client, UDP工作模式, 通过软件轻松配置。适合各种需要网络操作的物联场景，USR-TCP232-200串口服务器如图2.1所示：



Fig. 2.1 USR-TCP232-200 Serial Server Schematic diagram

图 2.1 USR-TCP232-200 串口服务器示意图

## 2.2.Modbus工业控制通讯协议

Modbus工业控制协议是应用于电子控制器上的一种通用语言协议。通过此协议，控制器相互之间、控制器经由网络（例如以太网）和其它设备之间可以通信。事实上，它已经成为一通用工业标准。有了它，不同厂商生产的控制设备可以连成工业网络，进行集中监控。此协议定义了一个控制器能认识使用的消息结构，而不管它们是经过何种网络进行通信的。它描述了一控制器请求访问其它设备的过程，如何回应来自其它设备的请求，以及怎样侦测错误并记录。它制定了消息域格局和内容的公共格式。当在一Modbus网络上通信时，此协议决定了每个控制器须要知道它们的设备地址，识别按地址发来的消息，决定要产生何种行动。如果需要回应，控制器将生成反馈信息并用Modbus协议发出。



Fig. 2.2 USR-TCP232-200 Serial Server Communication Schematic diagram

图 2.2 USR-TCP232-200串口服务器通信示意图

## 2.3.系统开发环境及技术

### 2.3.1.Tomcat简介

Tomcat是由Apache软件基金会下属的Jakarta项目开发的一个Servlet容器，按照Sun Microsystems提供的技术规范，实现了对Servlet和JavaServer Page（JSP）的支持，并提供了作为Web服务器的一些特有功能，如Tomcat管理和控制平台、安全域管理和Tomcat阀等。由于Tomcat本身也内含了一个HTTP服务器，它也可以被视作一个单独的Web服务器。

### 2.3.2.MySql简介

我们知道MySQL由于优异的性能成为很流行的关系型数据库，而由于MySQL架构分为：服务器层和存储引擎层，且服务器层对外提供统一的接口访问，所以基础开发遵循接口规范即可，无需关心太多，而存储引擎由于有多种且各个引擎特性和功能不尽相同，又由于在大量的生产实践中，InnoDB和MyISAM表现出的优异性能，所以目前行业内使用的MySQL引擎主要就是InnoDB和MyISAM，因此我们在使用时需要了解MySQL主流的两种引擎InnoDB和MyISAM的联系和区别。InnoDB表数据组织在主键索引中，实际上主键索引树的叶子节点存储的即为真实表数据，且主键索引为聚簇索引，即数据都是按主键排序组织，连续存储，所以检索数据时按主键排序性能非常优异；对于MyISAM数据表，主键索引和表数据是分开组织的。表数据是按照行数据插入的顺序存储在磁盘上，即按照行号大小排序组织数据，所以MyISAM查询数据时底层是按照数据行的行号来检索的，这一点和InnoDB是很大的不同，也决定了MyISAM索引组织方式和InnoDB有很大的不同

### 2.3.3.ExtJS框架

ExtJS可以用来开发RIA也即富客户端的AJAX应用，是一个用javascript写的，主要用于创建前端用户界面，是一个与后台技术无关的前端ajax框架。因此，可以把ExtJS用在.Net、Java、Php等各种开发语言开发的应用中。ExtJs最开始基于YUI技术，由开发人员JackSlocum开发，通过参考JavaSwing等机制来组织可视化组件，无论从UI界面上CSS样式的应用，到数据解析上的异常处理，都可算是一款不可多得的JavaScript客户端技术的精品。

## 2.4.系统使用技术

### 2.4.1.SpringMVC框架

Spring MVC属于SpringFrameWork的后续产品，已经融合在Spring Web Flow里面。Spring 框架提供了构建 Web 应用程序的全功能 MVC 模块。使用 Spring 可插入的 MVC 架构，可以选择是使用内置的 Spring Web 框架还可以是 Struts 这样的 Web 框架。通过策略接口，Spring 框架是高度可配置的，而且包含多种视图技术，例如 JavaServer Pages（JSP）技术、Velocity、Tiles、iText 和 POI。Spring MVC 框架并不知道使用的视图，所以不会强迫您只使用 JSP 技术。Spring MVC 分离了控制器、模型对象、分派器以及处理程序对象的角色，这种分离让它们更容易进行定制。

# 需求分析

## 3.1.系统概述

在传统照明领域里的照明系统大多采用普通的供电方式和控制方式。不仅浪费电力资源，而且还无法做到随开随关，给实际的使用带来极大的不便。在信息化的浪潮下，当把物联网的概念引入到照明控制领域，则给照明领域带来了翻天覆地的变化。在本文叙述的智能楼宇灯控系统里，就是使用了物联网技术。本智能楼宇灯控系统使用集成ARM单片机的控制器进行指令控制，使用控制器上的RJ45接口进行供电，不再使用传统的电缆，不但提高了系统的安全性，还大大简化了布线的工作。在数据链路层控制器采用施耐德公司的Modbus协议与有人公司生产的USR-TCP232-200串口服务器进行通信，在应用层和网络层USR-232-200串口服务器使用TCP/IP协议和控制计算机进行通信，实现命令的发送和信息的反馈。因为使用了TCP/IP协议，理论上系统在一定程度上可以无限的拓展；同时，因为串口服务器的IP地址和服务端口可以通过技术手段自定义修改，所以系统也具有良好的可移植性。

## 3.2.系统目标

### 3.2.1.总体目标

(1)管理员可以通过该信息系统将某一楼宇的结构，串口服务器，控制器以及灯具信息录入系统，建立起系统的初始逻辑结构。

(2)当楼宇结构，串口服务器，控制器以及灯具信息改变时，管理员可以及时对系统进行参数修改。

(3)管理员和普通用户可以通过控制室的控制计算机对楼宇中的所有灯具进行远程控制，包括设置灯具的点亮模式和灯具的亮度。

(4)管理员和普通用户可以通过系统实时监控系统中存在的任何一盏灯具的点亮状态和点亮亮度，并通过图形化的界面显示出来。

### 3.2.2.功能目标

(1)权限管理：对具有系统超级权限的管理用户和一般权限的普通用户进行管理。

(2)楼宇管理：对系统的楼宇，楼栋，楼层，房间进行管理。

(3)系统管理：对系统的分配器（串口服务器）和灯具控制器进行管理。

(4)灯具管理：对系统中开关类型，开关，灯具类型，灯具，以及灯具和开关之间的关联关系进行管理。

(5)实时控制：对系统楼宇中的灯具进行状态设置以及查看这些灯具的状态。

### 3.2.3.性能目标

(1) 通过降低系统响应时间，让用户能快速的访问系统，用户对系统的操作能及时作出反馈；

(2) 系统在检索关键信息方面提供精确查询和模糊查询两种方式，精确查询需要用户提供完整的检索关键字，模糊查询允许用户只提供部分关键字，系统会查询出于关键字相关的信息并展示；

(3)系统具有良好的拓展能力，以便二次开发时能与原系统兼容；

(4)系统操作界面简单明了，易于操作，人性化好当用户输入错误用户名验证时，系统提示用户用户名输入错误；当用户输入检索关键字与系统要求不一致时，采用错误提醒机制，提示用户输入正确数据和正确的操作系统；

(5)系统应当具有容错处理机制，当用户进行错误访问或者外部进行攻击时，系统能及时识别，防止错误继续蔓延；

(6)系统应当具有99.9%的可用性，当系统所依赖的网络环境、硬件环境以及各个控制节点出现异常时，系统能及时发现并提供降级服务。

## 3.3.系统功能

### 3.3.1.全局功能模块

本次毕业设计开发的基于TCP/IP协议与Modbus工控协议的楼宇智能照明控制系统分为五个大模块。

第一个模块是权限管理模块，此模块又有权限管理子模块，主要实现的是对系统中所有的用户权限进行授权、解除、检索和修改操作；第二个模块是楼宇管理模块，它分为楼宇管理子模块、楼栋管理子模块、楼层管理子模块和房间管理子模块，这几个模块分别对楼宇、楼栋、楼层和房间进行录入、删除、检索查看和修改等管理工作，并且楼宇、楼栋、楼层和房间之间按照实际的结构组织起来，进行信息化管理；第三个模块是系统管理，分为分配器管理子摸块和控制器管理子模块，分配器管理子模块主要对实际布线环境中存在的分配器录入、检索和修改，当分配器出现故障需要更换时，再进行删除，除此之外还可对分配器的必须参数进行可配置化管理，控制器管理子模块是对系统中存在的控制器进行录入、删除、检索和修改；第四个模块是灯具管理模块，其下又包括灯具类型管理子模块、灯具管理子模块、开关类型管理子模块、开关管理子模块和关联设置子模块，前四个子模块分别对各自管理的实体进行增加、删除、检索和修改，最后一个子模块关联设置子模块是将灯具实体和开关实体进行统一管理，并将它们关联起来，建立联系；最后一个模块是实时控制模块，主要由灯具状态设置子模块和灯具状态监控子模块构成，这两个模块分别实现对录入系统的楼宇实际布线环境中的灯具进行点亮状态的设置以及灯具的状态实时进行监控。

此智能照明控制系统的功能模块图如下所示：



Fig. 3.1 Global function module diagram

图 3.1 全局功能模块图

### 3.3.2.灯控相关功能模块

本智能照明控制系统的用户端下辖三个模块，分别是楼宇管理模块、灯具管理模块和实时控制模块。楼宇模块分为楼宇管理子模块、楼栋管理子模块、楼层管理子模块和房间管理子模块，这些模块都是建立起系统初始化运行环境的接口，用户可以根据实际环境进行添加各个实体，删除各个实体，也可以查看先前添加的实体，当实际环境发生变化后，用户还可以进行重新修改配置；灯具管理模块下辖灯具类型管理子模块、灯具管理子模块、开关类型管理子模块、开关管理子模块和关联设置子模块，用户可以通过前四个子模块提供的接口对各个模块管理的实体进相关的管理工作，最后一个模块关联设置子模块是最重要的一个模块，利用这个模块，用户可以对系统中的灯具和开关进行统一的管理，将它们关联起来，建立联系；实时控制模块是管理员和普通用户都能使用的模块，此模块下面的灯具状态设置子模块负责对录入系统的灯具设置状态，而灯具状态监控子模块负责对灯具状态进行实时的监控，这两个子模块是此智能照明控制系统的最主要的功能模块。图3-3是用户端的系统功能模块图，如下图所示：



Fig. 3.2 Light control related function module diagram

图 3.2 灯控相关功能模块示意图

## 3.4.系统用例

### 3.4.1.系统全局用例

用例图是由参与者（Actor）、用例（Use Case）、箭头组成、系统边界，用画图的方法来完成。参与者（Actor）：表示程序或系统进行交互的用户、组织或外部系统。用一个小人来表示。用例（Use Case）：就是外部可见的系统功能，对系统提供的服务进行相应的描述。用椭圆来表示。用例的关系有：关联、泛化、包含以及扩展关系。由于本文不是对UML进行研究，所以此处就不再对UML中的用例图详细的展开叙述，本智能楼宇灯控系统全局用例如图3.3所示：



Fig. 3.3 System global use case diagram

图 3.3 系统全局用例图

用例说明：

上图所示的系统全局用例图的参与者是系统管理员。主要事件有管理权限、管理楼宇、管理系统、管理灯具和实时控制。当用户点击权限管理选项后，系统进入管理权限界面，在这个界面里用户可以进行系统用户的相关管理工作；用户点击楼宇管理选项后，系统向用户展示楼宇管理、楼栋管理、楼层管理和房间管理子选项，用户可以点击任意一个选项进入相应的管理界面，比如用户点击房间管理选项后，系统进入房间管理的界面，在这个界面用户可以添加新房间到系统中，可以删除与实际情况不符的房间信息，可以检索系统中已存在的房间信息展示出来，还可以对系统中存在的房间信息进行编辑修改，再次存入系统中；用户点击系统管理选项，系统显示分配器管理和控制器管理子选项，比如用户点击分配器管理选项后，系统进入分配器管理界面，在这个界面中又包含了录入分配器、解除无效的分配器、检索查看存在的分配器和重新配置分配器参数；用户点击灯具管理选项后，系统提供给用户的子选项有:灯具类型管理、灯具管理、开关类型管理、开关管理和关联配置子选项，比如用户点击灯具管理选项后，系统进入灯具管理界面，此界面有添加灯具、删除灯具、检索灯具和编辑灯具等接口，用户可以点击上述任一接口对灯具信息进行管理，又比如用户点击关联设置选项后，系统进入关联设置界面，在后台将系统中已存在的所有控制器显示出来，用户选择任一控制器点击，即可显示此控制器上端口的关联信息；用户点击实时控制选型后，系统给出用户灯具状态设置和灯具状态监控子选项，当用户点击灯具状态设置子选项后，系统进入灯具状态设置界面，在此界面显示系统中所有的灯具信息，用户可以选择一个或多个灯具进行状态设置，当用户点击灯具状态监控子选项后，系统进入灯具状态监控界面，在界面中显示系统中所有房间信息，用户可以选择其中任意一个房间，即可显示此房间中的所有灯具亮灭状态以及亮度状态。

### 3.4.2.关联设置用例

关联设置主要提供灯控系统中灯具实例、开关实例与控制器实例之间的关联关系建立功能。当客户端向某个节点（节点在此处以及下文中代指灯具和开关）发出了某个指令后，为了确保这条指令能准确的控制到某个节点，这里的“关联关系”功不可没。而“关联关系”的实质指的是灯具与开关的对应关系以及节点与控制器（端口）之间的对应关系，这里对应关系有可能是一对一、一对多、多对一和多对多。由于此部分主要说明的是关联设置的用例设计，故对其细节阐述不再进行，下面我们来看一下关联设置功能的用例图，如下图3.4所示：



Fig. 3.4 Associated settings use case diagram

图 3.4 关联设置用例图

用例描述：

1. 用例：查询关联信息

参与者：系统管理员

主事件流：

1. 管理员选择关联设置选项；
2. 照明控制系统做出响应，系统在后台查询录入系统的所有开关信息，页面跳转至关联设置界面，所有开关信息通过界面显示出来；
3. 管理员选择一个开关，查询其关联信息；
4. 系统接收将要查询关联信息的开关ID，并通过此ID查询此开关的地址信息，接下来利用开关的地址信息，转换成符合Modbus通信协议的查询命令，使用Java Socket通信发送给控制器，系统做好接收响应信息的准备，当系统接收到返回信息后，对信息进行加工处理解析其中的信息，在前台界面通过弹出框的形式展示给管理员用户查看，查询关联信息用例结束；
5. 用例：关联端口

参与者：系统管理员

主事件流：

1. 管理员选择关联设置选项；
2. 照明控制系统做出响应，系统在后台查询录入系统的所有开关信息，页面跳转至关联设置界面，所有开关信息通过界面显示出来；
3. 管理员选择一个开关，查询其关联信息；
4. 系统通过一系列操作查询出此开关的关联信息，写入数据库中，并通过前台界面显示出来；
5. 管理员根据实际情况，选择开关的关联情况，确认无误后提交保存；
6. 系统接收开关的的关联信息，通过一定的算法将关联信息转换成符合Modbus通讯协议的命令，使用Java Socket将命令发送给控制器，系统做好接受响应信息的准备，当系统接收到返回信息后，通过一定规则将信息解析，如果关联成功，则系统通过提示框给用户提示关联成功，如果不成功，则系统同样通过弹出提示框的形式提示用户关联不成功，关联端口用例结束；

### 3.4.3.实时控制用例

灯具状态监控模块是本系统的重要模块，它包括灯具状态设置子模块和灯具状态监控子模块，这两个子模块都是与灯具状态相关的模块，它们实现的功能是互逆的，即灯具状态设置是将灯具状态的一系列参数写入控制器，在通过控制器控制灯具状态，而灯具状态监控是将存储在控制器中的灯具状态信息通过读命令读出来，所以在设计用例的时候，虽然它们分属不同的子模块中，我还是将它们设计在了一起，如下图3.5所示就是灯具实时控制的用例图：



Fig. 3.5 Real - time control use case diagram

图 3.5 实时控制用例图

用例描述：

1. 用例：灯具状态设置

参与者：系统管理员和用户

主事件流：

1. 用户选择灯具状态设置选项；
2. 照明控制系统做出响应，系统在后台查询录入系统的所有灯具信息，页面跳转至灯具状态设置界面，灯具信息通过界面显示出来；
3. 用户选择一盏灯具或选择多盏灯具，点击设置状态按钮；
4. 系统弹出设置状态的弹出框；
5. 用户填写需要设置的一系列参数，确认无误后，点击提交设置按钮；
6. 系统接收用户填写的设置参数，若参数格式正确且数据合法，系统则予以提交参数进行设置，否则通过提示框提示用户灯具状态设置参数不正确，需重新填写；
7. 用户填写参数通过系统验证后，系统将参数转换成符合Modbus通讯协议的命令，使用Java Socket将命令发送给控制器，系统做好接收响应信息的准备，当系统接收到响应信息后，将响应信息解析，如果设置成功，系统通过提示框提示用户灯具状态设置成功，否则通过提示框提示用户，提示灯具状态设置失败，灯具状态设置用例结束；
8. 用例：灯具状态监控

参与者：系统管理员和用户

主事件流：

1. 用户选择灯具状态监控选项；
2. 照明控制系统做出响应，页面跳转至灯具状态监控界面，系统查询录入系统中的所有房间信息，并通过界面显示出来；
3. 用户选择需要查看灯具状态的房间，点击查看状态按钮；
4. 系统接收需要查看灯具状态房间的ID，根据此ID查询出属于此房间的灯具，再通过灯具自身的属性信息生成符合Modbus通讯协议的查询灯具状态的命令，使用Java Socket发送查询命令，系统做好接受响应信息的准备，当系统接收到响应信息后，系统则按照一定的规则对信息进行解析，最后得出此房间每盏灯的状态信息，并通过图形化界面显示出来；如果系统在等待一段时间后仍然没有接收到响应信息，系统则以提示框的形式给用户以提示，提示用户网络繁忙，查询失败，至此灯具状态用例结束；

# 数据库设计

根据用户对系统的需求，设计出系统的用例模型，再根据用例模型设计系统的各个实体之间的关系，然后再导出满足系统各个功能的数据库关系模式，最后根据数据库关系模式，结合实际使用场景，选用不同的数据库（高并发大存储量安全性高可选用付费的企业级Oracle、开发敏捷可跨平台学习成本低可选用开源的MySql、在特定的平台如Windows平台可使用SQL Server、响应迅速数据量少允许丢失可使用基于内存的Key-Value数据库Redis），并根据选中的数据库支持的类型，进行数据库表结构的设计。

## 4.1.数据库关系模式

通过前期对系统的需求分析，得到了用户对系统的期望也导出了系统的初步架构，在此基础上我们得出了系统数据库设计的关系模式以及范式说明（注：只列出了本人负责模块相关的数据表关系模式）：

1. 分配器（分配器编号，楼层编号，分配器名称，分配器IP地址，分配器服务端口）

主键：分配器编号

函数依赖{分配器编号->其他所有属性}

由函数依赖可知，依赖左端为候选码，所以分配器关系模式属于BCNF

1. 控制器（控制器编号，分配器编号，控制器编码，控制器地址码）

主键：控制器编号

函数依赖{控制器编号->其他所有属性}

由函数依赖可知，依赖左端为候选码，所以控制器关系模式属于BCNF

1. 灯具类型（灯具类型编号，类型名称，灯具品牌，灯具功率，灯具规格，灯具点亮图例，灯具熄灭图例）

主键：灯具类型编号

函数依赖{灯具类型编号->其他所有属性}

由函数依赖可知，依赖左端为候选码，所以灯具类型关系模式属于BCNF

1. 灯具（灯具编号，灯具类型编号，房间编号，开关编号，控制器编号，控制器端口号，灯具编码）

主键：灯具编号

函数依赖{灯具编号->其他所有属性}

由函数依赖可知，依赖左端为候选码，所以灯具关系模式属于BCNF

1. 开关类型（开关类型编号，开关类型名称，命令码）

主键：开关类型编号

函数依赖{开关类型编号->其他所有属性}

由函数依赖可知，依赖左端为候选码，所以开关类型关系模式属于BCNF

1. 开关（开关编号，开关类型编号，控制器编号，控制器端口，房间编号，开关名称，端口1状态，端口2状态…）

主键：开关编号

函数依赖{开关编号->其他所有属性}

由函数依赖可知，依赖左端为候选码，所以开关关系模式属于BCNF

## 4.2.数据库表结构

在需求分析的基础上我们得到了系统的用例模型，参考系统的功能需求、性能需求，在保证一定程度节省存储空间的同时，为了增删查改的高效，以及系统迁移扩展的方便，我们保留了一定的雍余设计。下面即是我们在此次智能楼宇灯控系统实施开发过程中设计的数据库表结构（注：只列出了本人负责模块相关的数据表结构）：

1.分配器表（serial\_server）

表 4.1 分配器表结构

Table 4.1 Serial\_Server table structure

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名称** | | **含义** | **数据类型** | **约束** |
| **ID** | | 大厦编号 | int | 主键(非空,自增) |
| **EDIFICENAME** | 大厦名称 | | varchar | 无 |
| **EDIFICEFADDRESS** | | 大厦地址 | varchar | 无 |

2.控制器表（controller）

表 4.2 控制器表结构

Table 4.2 Controller table structure

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名称** | **含义** | **数据类型** | **约束** |
| **ID** | 分配器编号 | int | 主键(非空,自增) |
| **STOREYID** | 楼层编号 | int | 外键(非空) |
| **SERVERCODE** | 分配器号 | varchar | 无 |
| **IPADDRESS** | 分配器IP地址 | varchar | 无 |
| **PORT** | 分配器服务端口 | varchar | 无 |

3.灯具类型表（lamp\_type）

表 4.3 灯具类型表结构

Table 4.3 Lamp\_Type table structure

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名称** | **含义** | **数据类型** | **约束** |
| **ID** | 灯具类型编号 | int | 主键(非空,自增) |
| **TYPENAME** | 灯具类型名称 | varchar | 无 |
| **BRAND** | 灯具品牌 | varchar | 无 |
| **POWER** | 灯具功率 | double | 无 |
| **LAMPSPECICATION** | 灯具规格 | varchar | 无 |
| **LAMPTYEPICTUREON** | 灯具点亮照片 | varchar | 无 |
| **LAMPTYPICTUREOFF** | 灯具熄灭照片 | varchar | 无 |

4.灯具表（lamp）

表 4.4 灯具表结构

Table 4.4 Lamp table structure

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名称** | **含义** | **数据类型** | **约束** |
| **ID** | 房间编号 | int | 主键(非空,自增) |
| **STOREYID** | 楼层编号 | int | 外键(非空) |
| **ROOMCODE** | 房间号 | varchar | 无 |
| **ROOMAREA** | 房间面积 | double | 无 |
| **ROOMPICTURE** | 房间照片 | varchar | 无 |

5.开关类型表（switch\_type）

表 4.5 开关类型表结构

Table 4.5 Switch\_Type table structure

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名称** | **含义** | **数据类型** | **约束** |
| **ID** | 开关类型编号 | int | 主键(非空,自增) |
| **SWITCHTYPENAME** | 开关类型名称 | varchar | 无 |
| **COMMANDCODE** | 命令码 | Varchar | 无 |

6.开关表（switch）

表 4.6 开关表结构

Table 4.6 Switch table structure

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名称** | **含义** | **数据类型** | **约束** |
| **ID** | 开关编号 | int | 主键(非空,自增) |
| **SWITCHTYPEID** | 开关类型编号 | int | 外键(非空) |
| **CONTROLLERID** | 控制器编号 | int | 外键(非空) |
| **CONTROLLERPORT** | 控制器端口 | varchar | 无 |
| **ROOMID** | 房间编号 | int | 外键(非空) |
| **SWITCHCODE** | 开关号 | varchar | 无 |
| **P1** | 端口1状态 | int | 无 |

## 4.3.系统类图设计

类图(Class diagram)是显示了模型的静态结构，特别是模型中存在的类、类的内部结构以及它们与其他类的关系等。类图不显示暂时性信息。类图(Class diagram)由许多（静态）说明性的模型元素（例如类、包和它们之间的关系，这些元素和它们的内容互相连接）组成，类图可以组织在（并且属于）包中，仅显示特定包中的相关内容。

在软件开发的不同阶段使用的类图具有不同的抽象层次。一般类图可分为3个层次，即概念层、说明层和实现层，把类图划分为3个层次对于画类图或阅读类图非常有用。

根据前期的分析设计，我得到了本系统的实体类图（注：该类图为了说明整个系统类之间的关系，故包含整个系统的所有类图），如下图4.1所示：



Fig. 4.1 System class diagram

图 4.1 系统类图

# 系统设计

## 5.1.系统架构

该基于TCP/IP协议与Modbus通讯协议的智能楼宇灯控系统由于既涉及传统WEB应用又涉及最近火热的物联网，实现起来有一定难度。通过调研学术资料以及参考业内通行做法，我设计如下系统架构。首先该系统是建立在WEB应用的基础上，WEB架构为B/S架构，B端可以是普通PC、智能Phone、便携式Pad以及能使用Browser的各种终端，服务端由于目前单机并发量处在可控的范围内，故1.0版本并没有使用多机器集群，若在系统迭代过程中，系统并发量快速增长，且单机系统遇到瓶颈，可考虑使用LVS、nginx等负载均衡技术配合多机器集群，以解决此问题。该系统WEB应用部分充当整个系统的View和Controller，View是指系统给用户提供统一的可视化视图，Controller指的是所有控制命令都是在WEB应用中生成，然后通过网络发送出去。WEB应用与串口服务器（也叫分配器，下文对出现的两种叫法不予区分）的通讯是通过TCP/IP网络协议以字节流的形式完成的。当基于Modbus工控协议的命令被以字节流的形式发送到串口服务器，串口服务器会以广播的形式将命令发送给与该串口服务器互联的所有控制器，由于基于Modbus协议的命令中包含了控制器的地址码，所以当各个控制器将命令解析后地址码与自身符合的才会进一步处理。控制器内部含有一个AT89C51单片机，会完全的将命令的含义解析出来，通过对端口输出不同的电压进而直接控制连在这些端口上的灯具或者通过直接控制连在这些端口上的开关间接控制与这些开关互联的灯具。为了更加直观的说明这种控制关系，系统的整体架构图如下图5.1所示：

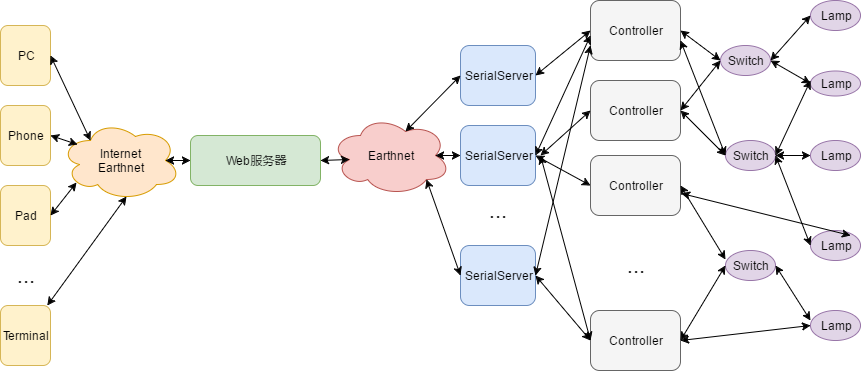


Fig. 5.1 System architecture diagram

图 5.1 系统架构图

## 5.2.WEB系统设计

本智能照明控制系统在设计过程中考虑到系统的可拓展性，并没有使用传统的JavaEE Servlet，而采用了Spring MVC 框架技术，符合流行的MVC（Model-View-Controller）三层设计。其中Model层主要用于数据的存储，以及数据在各个控制层之间流通时状态的保存，如果考虑到解耦性和普适性，我们可以给Model层细分为View视图对象，Model模型对象，Param参数对象，以及DAO数据库域对象；View层主要职责是将后端数据的一系列变化通过一定的UI技术展示出来，现阶段可用的技术有Html、Jsp、FreeMaker技术等，同时还可使用JavaScript脚本语言及其相关框架例如Jquery、Angular JS、node.js等；Controller层主要是后台系统的控制中枢，负责接收客户端发送的请求，处理客户端请求中的数据，并对Model层中的数据进行增删查改操作，然后再将数据的修改持久化到数据存储介质中（数据库、文件系统、磁盘、磁带等），待完成这些工作后，再将数据封装在View对象中通过View层展示给客户端。

在经典软件系统开发过程中，我们一般都会在控制层和数据存储层之间加上一层，这层一般叫做中间层或者组件层，JavaEE中著名的EJB就是组件层的代表产物。这样做的好处是显而易见的：控制层在接受客户端请求后，一般都会对数据库进行某种操作，如果没有中间层（组件层），那我们一般会在控制层直接操作数据库，这样不是不可以，只是控制层和数据存储层完全的耦合在了一起，如果数据存储层做出修改，那么控制层也要做出修改，这显然是不可接受的，但是，有了中间层后控制层不再关心数据存储层的变化，控制层只需要通过组件层对数据存储层进行操作，这些操作对控制层是完全透明的，这就降低了系统各个层之间的耦合性，也符合软件工程中对软件项目推荐的高内聚，低耦合。下图5.2是MVC设计模式交互示意图：

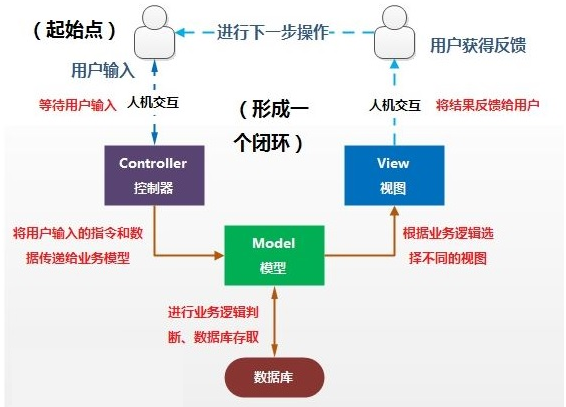


Fig. 5.2 MVC Design pattern diagram

图 5.2 MVC 设计模式示意图

## 5.3.系统功能时序设计

### 5.3.1.关联设置时序图

表 5.1 关联设置时序图描述

Table 5.1 Associative set timing diagram description

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 描述 |
| main.jsp | Jsp页面 | 显示数据信息，与用户交互 |
| SwitchCtrl | 控制类 | 接收用户请求返回响应信息 |
| SwitchManager | 业务实现类 | 基本业务实现类 |
| SwitchDao | 数据库操作类 | 基本数据库操作类 |
| ConfigureSwitchModelManager | 业务实现类 | 基本业务实现类 |
| SocketService | 普通java类 | 发送通信命令类 |
| DataBase | 数据库 | 用户存放数据 |



Fig. 5.3 Association set timing diagram

图 5.3 关联设置时序图

关联设置时序图说明：

当用户进入灯具管理模块并选择关联设置子模块后，即可点击关联设置按钮进行关联设置操作。用户点击关联设置按钮向系统发出关联设置命令，照明控制系统做出响应。第一阶段开始，main.jsp页面向SwitchCtrl发送Ajax请求并传递参数，控制层的SwitchCtrl调用业务逻辑层SwitchManager中的getSwitchInfoById()方法查询需要关联端口的开关信息，业务逻辑层SwitchManager再调用数据层SwitchDao中的getSwitchInfoById()方法，最终数据层SwitchDao使用jdbc的方式查询数据库中开关信息并将开关信息封装成JavaBean返回至控制层SwitchCtrl,第一阶段结束；第二阶段开始，控制层调用业务层CheckSwitchModelManager中的handle()方法查询开关的已关联信息，业务层CheckSwitchModelManager再调用SocketService中的receiveResult()方法使用Java Sokect通信向控制器发送查询命令，系统做好接收响应信息准备，一旦接收到响应信息，则进行处理解析，并将响应的关联信息封装在JavaBean中返回至控制层SwitchCtrl，控制层SwitchCtrl再给main.jsp界面发送通知，main.jsp界面通过面板显示已关联信息，第二阶段结束；第三阶段开始，用户修改编辑开关端口关联情况提交设置，main.jsp向控制层SwitchCtrl发出请求传递关联信息参数，控制层SwitchCtrl将关联信息封装在JavaBean里，紧接着控制层SwitchCtrl调用业务逻辑层ConfigureSwitchModelManager中的handle()方法关联端口，业务逻辑层ConfigureSwitchModelManager再调用SocketService中的sendCommand()方法，系统做好接收控制器返回的状态信息准备，当接收到返回状态信息后，立即进行解析并将解析后的信息返回至控制层SwitchCtrl,控制层SwitchCtrl再将状态信息响应给main.jsp界面，最后main.jsp界面利用弹出框通知用户开关端口关联设置成功或者失败，第三阶段结束。至此开关关联设置过程结束。

### 5.3.2.灯具状态监控时序图



Fig. 5.4 Lamp condition monitoring timing diagram

图 5.4 灯具状态监控时序图

表 5.2 灯具状态监控时序图描述

Table 5.2 Lamp condition monitoring timing diagram description

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 描述 |
| main.jsp | Jsp页面 | 显示数据信息，与用户交互 |
| LampCtrl | 控制类 | 接收用户请求返回响应信息 |
| LampManager | 业务实现类 | 基本业务实现类 |
| LampDao | 数据库操作类 | 基本数据库操作类 |
| CheckLampStatusModelManager | 业务实现类 | 基本业务实现类 |
| SocketService | 普通java类 | 发送通信命令类 |
| DataBase | 数据库 | 用户存放数据 |

灯具状态监控时序图说明：

当用户进入实时控制模块并选择灯具状态监控子模块后，即可点击灯具状态监控按钮进行灯具状态监控操作。用户点击灯具状态监控按钮向系统发出命令，照明控制系统对此做出响应。第一阶段开始，main.jsp以Ajax的方式向控制层LampCtrl发出请求并传递参数，控制层的LampCtrl调用业务逻辑层LampManager中的getLampListByRoomId()方法查询需要监控状态的灯具信息，业务逻辑层LampManager再调用数据层LampDao中的getLampListByRoomId ()方法，最终数据层LampDao使用jdbc的方式查询数据库中灯具信息并将灯具信息封装成JavaBean返回至控制层LampCtrl,第一阶段结束；第二阶段开始，控制层LampCtrl调用业务逻辑层CheckLampStatusManager的handle()方法查询灯具状态，业务逻辑层CheckLampStatusManager再调用SocketService中的receiveResult()方法向控制器发送命令，系统做好接收响应信息的准备，当系统接收到响应信息后，则对响应信息进行解析，解析完成后将解析后的信息返回至控制层LampCtrl中，在控制层LampCtrl中灯具信息被封装到JavaBean里，接下来，再调用业务逻辑层LampManager的convertLampToJSONObject()方法把JavaBean转换成JSONObjet对象，使用HttpResponse的输出流将灯具状态信息返回到main.jsp页面，最后main.jsp页面将灯具状态信息以图形化的界面显示实时的灯具状态信息，第二阶段结束，至此灯具状态监控过程结束。

# 系统测试

## 6.1.系统测试

### 6.1.1.测试方法

软件测试分为黑盒测试和白盒测试两种。

黑盒测试是以用户的角度，从输入数据与输出数据的对应关系出发进行测试。如果外部特性本身有问题或规格说明的规定有误，黑盒测试方法就是无法发现问题。

白盒测试软件产品的内部结构和处理过程，而不测试软件产品的功能，用于纠正软件系统在描述、表示和规格上的错误。

### 6.1.2.测试步骤

在实现组将验证所开发的程序后，交至测试组，由测试组的相关工作人进行测试，测试一般有以下几个步骤：

1．测试人员要仔细阅读有关资料，然后写测试的计划，测试用例，为测试做充足的准备。

2．代码会审：代码会审是由一组人通过阅读、讨论和争议对程序进行静态分析的过程。

3．单元测试： 单元测试集中在检查软件设计的最小单位-模块上，通过测试发现实现该模块的实际功能与定义该模块的功能说明不符合的情况，以及编码的错误。

4．集成测试：集成测试是将模块按照设计要求组装起来，同时进行测试，主要目标是发现与接口有关的问题。

5．验收测试：验收测试的目的是向未来的用户表明，系统能够像预定要求那样工作。

如果严格按照上述测试过程对信息系统进行了全面的测试，那么至少可以说明系统是符合要求的（仍然会存在bug，但目前不影响使用），那么下一步就可以编写软件使用说明文档与软件一同交给用户使用。

## 6.2.测试过程

### 6.2.1.登陆模块测试

登录模块的测试通过设计用户名和密码的测试用例，采用黑盒测试与白盒测试，对系统的登录模块进行测试。黑盒测试采用了等价类划分法，有效等价类和无效等价类如表6.1 所示。测试用例1-6 是有效等价类，测试用例7 是无效等价类。

表6.1 登陆模块测试用例

Table 6.1 Login module test case

|  |  |  |  |
| --- | --- | --- | --- |
| **编号** | **测试数据** | **预期结果** | **效果截图** |
| 1 | 用户名：空  密 码：空 | 提示：登陆失败，请确认登录用户和密码！ | C:\Users\绿小红\AppData\Local\Temp\ksohtml\wps20B4.tmp.jpg |
| 2 | 用户名：空  密 码：1111 | 提示：登陆失败，请确认登录用户和密码！ | C:\Users\绿小红\AppData\Local\Temp\ksohtml\wps20C5.tmp.jpg |
| 3 | 用户名：admin  密 码：空 | 提示：登陆失败，请确认登录用户和密码！ | C:\Users\绿小红\AppData\Local\Temp\ksohtml\wps20C6.tmp.jpg |
| 4 | 用户名：admi  密 码：1111 | 提示：登陆失败，请确认登录用户和密码！ | C:\Users\绿小红\AppData\Local\Temp\ksohtml\wps20D6.tmp.jpg |
| 5 | 用户名：admin  密 码：111 | 提示：登陆失败，请确认登录用户和密码！ | C:\Users\绿小红\AppData\Local\Temp\ksohtml\wps20E7.tmp.jpg |
| 6 | 用户名:admi  密 码：111 | 提示：登陆失败，请确认登录用户和密码！ | C:\Users\绿小红\AppData\Local\Temp\ksohtml\wps20F8.tmp.png |
| 7 | 用户名:admin  密 码：1111 | 登陆成功 | C:\Users\绿小红\AppData\Local\Temp\ksohtml\wps2108.tmp.jpgC:\Users\绿小红\AppData\Local\Temp\ksohtml\wps2109.tmp.jpg |

### 6.2.2.关联设置测试

关联设置功能模块中可以变化的变量只有控制器端口值，而且输入参数是以复选框的方式呈现的，所以在测试过程中我仅按正常的操作步骤执行，观察系统是是否出现超出预期的异常以及是否能够正确的关联端口。在测试过程中，第一步，先选中关联端口图标，观察是否出现关联端口提示；第二步，点击关联端口图标，观察系统是否弹出面板，且面板中显示此开关已关联端口的信息；第三步，重新对关联端口做出修改，提交保存，观察系统能否检测到关联端口信息已被修改且将关联信息同步到控制器和数据库中，至此测试过程结束。通过这三个步骤，此功能被证明是成功的，测试过程和测试效果如下表6.2所示：

表 6.2 关联设置测试用例

Table 6.2 Associate settings test cases

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 测试用例 | 预期结果 | 效果截图 |
| 1 | 选择关联端口图标 | 显示关联端口 | C:\Users\绿小红\AppData\Local\Temp\ksohtml\wps4B91.tmp.jpg |
| 2 | 点击关联端口选项 | 弹出面板，显示已关联端口信息 | C:\Users\绿小红\AppData\Local\Temp\ksohtml\wps4BA2.tmp.jpg |
| 3 | 选择需要关联的端口，点击保存 | 提示：开关关联设置成功！ | C:\Users\绿小红\AppData\Local\Temp\ksohtml\wps4BB3.tmp.jpg |

### 6.2.3.灯具状态设置测试

灯具状态设置是本照明控制系统的重中之重，但是由于此功能效果只是将灯具置亮或者熄灭，所以无法进行大规模测试。为了测试当用户没有选择灯具、选择一盏灯具、选择多盏灯具和选择所有灯具时系统能否正常的做出响应，我设计了如下的测试用例。当用户没有选择灯具时，点击设置灯具状态系统应该弹出提示框，请选择要设置状态的灯具；当用户选择一盏灯、多盏灯、甚至所有灯时，系统应弹出设置灯具状态面板，且面板里能显示用户选中了多少灯具，用户此时可以设置灯具的亮灭状态，亮度数值，是否获取网络控制权限等，确认无误后提交设置，系统对此做出响应，将一系列灯具状态参数发送出去，实际使用中，被控灯具将会按照用户设置的灯具状态参数改变自身状态，完成这些后，系统再以弹出框的形式，提示用户灯具状态设置完成。测试过程和测试效果如下所示：

表 6.3 灯具状态设置测试用例

Table 6.3 Lamp Status Set Test Case

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 测试用例 | 预期结果 | 效果截图 |
| 1 | 不选择灯具号 | 提示：请选择要设置状态的灯具 | C:\Users\绿小红\AppData\Local\Temp\ksohtml\wps79C3.tmp.jpg  C:\Users\绿小红\AppData\Local\Temp\ksohtml\wps79D3.tmp.jpg |
| 2 | 选择一盏灯具 | 提示：灯具状态设置成功 | C:\Users\绿小红\AppData\Local\Temp\ksohtml\wps79E4.tmp.jpg  C:\Users\绿小红\AppData\Local\Temp\ksohtml\wps79E5.tmp.jpg |
| 3 | 选择5盏灯具 | 提示：灯具状态设置成功 | C:\Users\绿小红\AppData\Local\Temp\ksohtml\wps79F5.tmp.jpg  C:\Users\绿小红\AppData\Local\Temp\ksohtml\wps7A06.tmp.jpg |
| 4 | 选择所有灯具 | 提示：灯具状态设置成功 | C:\Users\绿小红\AppData\Local\Temp\ksohtml\wps7A07.tmp.jpg  C:\Users\绿小红\AppData\Local\Temp\ksohtml\wps7A18.tmp.jpg |

### 6.2.4.灯具状态监控测试

1. 测试用例：监控经理办公室

预期结果：1盏灯，且为熄灭状态

效果截图：

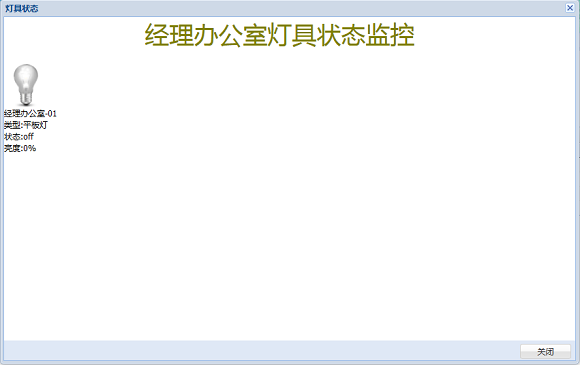


Fig. 6.1 Manager 's office lighting status monitoring diagram

图6.1经理办公室灯具状态监控示意图

1. 测试用例：监控会议室

预期结果：14盏灯，且3盏为熄灭状态，11盏为点亮状态

效果截图：



Fig. 6.2 Conference room lighting status monitoring diagram

图 6.2会议室灯具状态监控示意图

1. 测试用例：监控走廊

预期结果：3盏灯，1盏灯为熄灭状态，2盏灯为点亮状态

效果截图：



Fig. 6.3 Corridor lighting state monitoring diagram

图 6.3走廊灯具状态监控示意图

# 系统运行效果

在经过历时3个月的开发工作后，本智能照明控制系统编码工作完成。在接下来的测试过程中发现的一些问题在后续的维护过程中也一一完善，最终系统开发工作圆满完成，没有发现明显的bug，很好的满足了系统设计阶段的预期需求。本章主要将对智能照明控制系统的运行效果一一展示，让读者对本系统有一个更深更全面的了解。

## 7.1.系统全局效果

### 7.1.1.系统登陆界面

本智能照明控制系统的使用者角色分为系统管理员和普通用户。系统管理员主要在系统全局范围进行操作，负责建立、修改、删除和维护系统的运行环境，他有权添加、删除、修改系统的用户（系统管理员和普通用户），系统所具有的所有功能，管理员都可以参与，所以他的权限是最大的；普通用户角色主要权限是维护系统运行环境，查看系统的各项环境参数，使用系统的实时控制模块对录入系统的灯具进行状态设置和状态监控。为了控制这两个角色的操作行为，必须编写登陆模块，以便在用户界面接口层级上对他们进行隔离，确保系统运行的安全性和数据的安全性。下图是本系统的登陆主界面：



Fig. 7.1 Login home page effect diagram

图 7.1 登录主页效果示意图

### 7.1.2.系统菜单结构

系统用户键入用户名和密码确认登陆后，系统首先验证用户名、密码的合法性，如果用户名或密码与数据库中的数据不一致甚至不存在，系统则提示用户名和密码不匹配；如果用户名和密码正确，系统再判断用户的角色，并根据用户角色显示相应的界面和菜单。下图是管理员登陆成功后系统菜单结构：



Fig. 7.2 System menu structure sh schematic

图 7.2 系统菜单结构示意图

### 7.1.3.系统主界面

系统管理员或者普通用户在登陆成功后，系统查询数据库中的菜单信息，并按照一定的规则组织起来，数据流从控制层流向视图层，页面从登陆页转向主界面。系统的主界面效果如下所示：

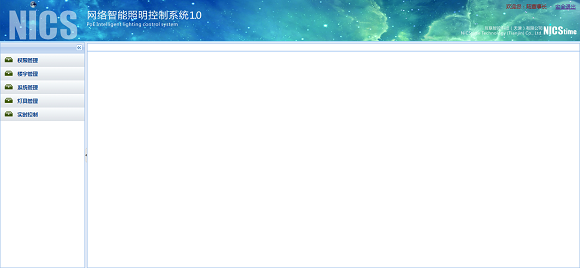


Fig. 7.3 System main interface diagram

图 7.3 系统主界面示意图

## 7.2.系统功能模块效果

### 7.2.1.关联设置

1.关联设置主界面

在灯具管理模块中，包含关联设置模块。此模块是将开关所在的控制器端口和灯具所在的控制器端口关联起来。关联设置主界面如下所示：



Fig. 7.4 Associate the main interface diagram

图 7.4 关联设置主界面示意图

2.关联设置面板

当用户选中开关并点击关联设置按钮后，系统查询此开关的相关属性信息和已关联端口信息，并通过面板显示出来。开关关联设置面板如下所示：



Fig. 7.5 Associated settings panel diagram

图 7.5 关联设置面板示意图

当关联设置操作成功后，界面给出提示如下图7.6所示：

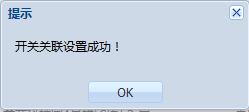


Fig. 7.6 The association set operation is successful

图 7.6 关联设置操作成功示意图

### 7.2.2.灯具状态设置

1.灯具状态设置主界面

灯具状态设置模块主要负责对录入系统的所有灯具状态进行设置，当用户选择进入灯具状态设置模块后，页面跳转至灯具状态设置页面：

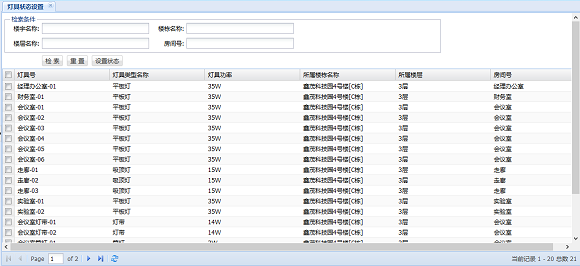


Fig. 7.7 Lamp state setting main interface diagram

图 7.7灯具状态设置主界面示意图

2.灯具状态设置面板

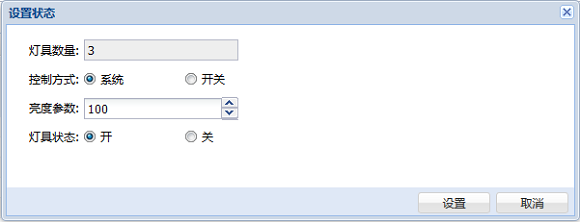


Fig. 7.8 Lamp state setting panel diagram

图 7.8 灯具状态设置面板示意图

接着上一步，用户选择了三盏灯具，并选中了控制模式为系统，即系统获取网络控制权限，亮度参数为100（1-100范围），灯具状态为开状态，点击设置按钮，系统会在后台将这些灯具状态信息按照一定的规则转换为符合Modbus通讯格式的命令，使用Java Socke发送给控制器，在经历短暂延迟后，控制器做出反应，控制实际的灯具做出应有的反应，此时，系统会以弹出框的形式给出设置状态成功的提示：



Fig. 7.9 Lamp state setting is successful

图7.9灯具状态设置成功示意图

### 7.2.3.灯具状态监控

1.灯具状态监控主界面

灯具状态监控模块主要实现对录入系统的所有灯具状态进行实时的监控，当用户选择进入灯具状态监控模块后，系统即在后台查询所有灯具信息，并按照房间来进行分组显示出来，用户在状态监控主界面可以按房间信息检索所有灯具，也可以选择其中某个房间，点击查询灯具状态按钮，监控此房间里的所有灯具实时状态信息。灯具状态监控主界面如下所示：

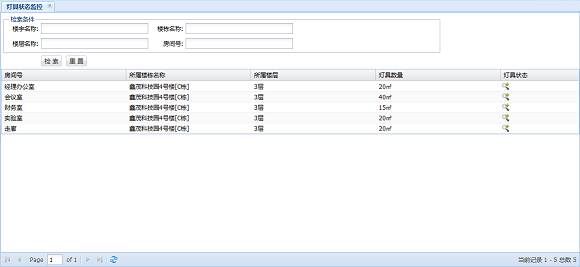


Fig. 7.10 Lamp state monitoring main interface

图 7.10 灯具状态监控主界面

2.灯具状态监控面板

在上一步的操作中，用户给系统的部分灯具设置了点亮的状态，在灯具状态监控模块里即可对这些灯具状态进行监控。当用户选择某个房间后，点击灯具状态监控按钮，系统会查询此房间里的所有灯具状态信息，并以图形化的形式显示出来。



Fig. 7.11 Lamp status monitoring panel diagram

图 7.11 灯具状态监控面板示意图

# 关键代码

一个好的系统离不开好的系统功能设计，而好的系统功能设计又依赖好的编码设计。在本次智能照明控制系统的开发过程中，为了实现部分功能，我进行了详细的代码设计过程，期间也编写了一些关键性的代码，在此，由于篇幅有限不可能一一列举出来，因此我选择了一些具有代表性的代码进行说明。

## 8.1.上传灯图例关键代码

上传灯图例功能是在灯具类型管理模块中，当我们在系统录入某个新的灯图例类型时，我们可以选择将该类型的灯图例照片录入系统，当我们在灯具状态监控功能中，查看某个房间的所有灯具状态时，显示出的灯具状态图是以灯图例的照片为显示依据。下面是上传灯图例关键代码：

清单 8.1 上传灯图例关键代码

public String uploadLampTypePicture() throws Exception{

LampType lampType = new LampType();

lampType.setId(Integer.parseInt(this.id));

ServletContext sc = ServletActionContext.getServletContext();

ApplicationContext ac2 = WebApplicationContextUtils.getWebApplicationContext(sc);

LampTypeManager lampTypeManager = (LampTypeManager) ac2.getBean("lampTypeManager");

try{

String pathOn=ServletActionContext.getServletContext().getRealPath("/lamppicture/on");

String pathOff=ServletActionContext.getServletContext().getRealPath("/lamppicture/off");

File filePathOn=new File(pathOn);

File filePathOff=new File(pathOff);

if(!filePathOn.exists()){

filePathOn.mkdirs();

}

if(!filePathOff.exists()){

filePathOff.mkdirs();

}

for(int i=0;i<files.size();i++){

String fileName=filesFileName.get(i);

int index=fileName.lastIndexOf(".");

String suffix=fileName.substring(index);

String newFileName="";

if(i==0){

newFileName="lamptypepicture"+lampType.getId()+"\_on"+suffix;

FileUtils.copyFile(files.get(i), new File(filePathOn,newFileName));

lampType.setLamptypepictureon(newFileName);

lampTypeManager.editLampTypeById(lampType);

}

if(i==1){

newFileName="lamptypepicture"+lampType.getId()+"\_off"+suffix;

FileUtils.copyFile(files.get(i), new File(filePathOff,newFileName));

lampType.setLamptypepictureoff(newFileName);

lampTypeManager.editLampTypeById(lampType);

}

}

}catch(Throwable t){

t.printStackTrace();

}

return SUCCESS;

}

## 8.2.关联设置关键代码

关联设置在整个楼宇灯控系统中相当于枢纽功能，它的功能主要就是将我们在灯具管理和开关管理中录入系统的所有灯具开关与控制器管理中录入系统的所有控制器的端口之间建立关联关系，当我们对系统中某个节点发出控制指令时，控制命令首先被发送到控制器互联的串口服务器，串口服务器再将命令发送到控制器，控制器通过对命令码的解析得到命令语义，再通过向控制器的某个（几个）端口输出高低电平来达到控制效果。下面即是关联设置的关键代码：

清单 8.2 关联设置关键代码

public static boolean handle(Switch Switch){

int[] command=new int[9];

String strCommand="";

command[0]=Integer.parseInt(Switch.getAddresscode(),16);

command[1]=16;

command[2]=Integer.parseInt("1"+Switch.getControllerport(), 16);

command[3]=Integer.parseInt(Switch.getCommandcode());

command[4]=Switch.getCommanddata1();

command[5]=Switch.getCommanddata2();

command[6]=Integer.parseInt(Switch.getParameter());

int[] crc=CRCUtil.CRC16(command);

command[7]=crc[0];

command[8]=crc[1];

String space=" ";

char temp='0';

for(int i=0;i<command.length;i++){

if(i==8){

space="";

}

if(command[i]<=15){

strCommand+="0"+(Integer.toHexString(command[i]))+space;

continue;

}

strCommand+=(Integer.toHexString(command[i]))+space;

}

Boolean flag=SocketService.sendCommand(Switch.getIpaddress(), Integer.parseInt(Switch.getPort()), command);

return flag;

}

## 8.3.设置灯具状态关键代码

设置灯具状态是系统最主要的功能之一，主要是通过点击WEB界面按钮控制灯具的亮灭状态以及灯具的亮度，在这个过程中，最复杂也最核心的即是设置灯具状态的命令发送，由于Modbus协议的局限性，我在这里使用了字节流进行命令的传输，下面是设置灯具状态的关键代码：

清单 8.3 设置灯具状态关键代码

public static boolean handle(Lamp lamp){

int[] command=new int[9];

String strCommand="";

command[0]=Integer.parseInt(lamp.getAddresscode(),16);

command[1]=16;

command[2]=Integer.parseInt("4"+lamp.getControllerport(), 16);

command[3]=Integer.parseInt(lamp.getStatus());

command[4]=Integer.parseInt(lamp.getBrightness());

command[5]=0;

command[6]=Integer.parseInt(lamp.getNetauthority());

int[] crc=CRCUtil.CRC16(command);

command[7]=crc[0];

command[8]=crc[1];

String space=" ";

for(int i=0;i<command.length;i++){

if(i==8){

space="";

}

if(command[i]<=15){

strCommand+="0"+Integer.toHexString(command[i])+space;

continue;

}

strCommand+=Integer.toHexString(command[i])+space;

}

System.out.println(strCommand);

boolean flag=SocketService.sendCommand(lamp.getIpaddress(), Integer.parseInt(lamp.getPort()), command);

return flag;

}

public static boolean sendCommand(String ip,int port,int[] command){

boolean flag=true;

try {

Socket ss=new Socket();

ss.connect(new InetSocketAddress(ip,port), 1000);

ss.setSoTimeout(2000);

OutputStream out=ss.getOutputStream();

out.write(command[0]);

out.write(command[1]);

out.write(command[2]);

out.write(command[3]);

out.write(command[4]);

out.write(command[5]);

out.write(command[6]);

out.write(command[7]);

out.write(command[8]);

out.close();

ss.close();

Thread.sleep(200);

return flag;

} catch (IOException e) {

flag=false;

e.printStackTrace();

} finally{

return flag;

}

}

## 8.4.读取响应信息关键代码

与我们在WEB应用中常用的HTTP协议类似，Modbus协议也是基于request与response的通讯协议，当我们给控制器发送了一串命令，命令可能会（也可能不会）对节点状态产生影响，但不管是否产生了影响，控制器都会响应一串Modbus协议的命令，我们通过对命令的解析可以得到我们想要的信息，下面是读取响应信息的关键代码：

清单 8.4 读取响应信息关键代码

public static boolean receiveResult(String ip,int port,int[] command) {

boolean flag=true;

Socket ss=new Socket();

Socket sss = new Socket();

try {

ss.connect(new InetSocketAddress(ip,port), 1000);

ss.setSoTimeout(2000);

sss.setSoTimeout(2000);

OutputStream out=ss.getOutputStream();

out.write(command[0]);

out.write(command[1]);

out.write(command[2]);

out.write(command[3]);

out.write(command[4]);

out.write(command[5]);

out.write(command[6]);

out.write(command[7]);

out.write(command[8]);

out.close();

ss.close();

Thread.sleep(200);

sss.connect(new InetSocketAddress(ip, port), 1000);

sss.setSoTimeout(2000);

long start=System.currentTimeMillis();

int i = 0;

while (true) {

if (i > 8) {

break;

}

long time=System.currentTimeMillis()-start;

if(time>2000){

flag=false;

return flag;

}

if (sss.getInputStream().available() == 0) {

continue;

}

command[i++] = sss.getInputStream().read();

}

sss.getInputStream().close();

sss.close();

Thread.sleep(200);

return flag;

} catch (IOException e) {

flag=false;

e.printStackTrace();

} finally {

if(ss!=null&&ss.isConnected()){

try {

ss.close();

} catch (IOException e) {

e.printStackTrace();

}

ss=null;

}

if(sss!=null&&sss.isConnected()){

try {

sss.close();

} catch (IOException e) {

e.printStackTrace();

}

sss=null;

}

return flag;

}

}