毕业设计（论文）独创性声明

本人所呈交的毕业论文是在指导教师指导下进行的工作及取得的成果。除文中已经注明的内容外，本论文不包含其他个人已经发表或撰写过的研究成果。对本文的研究做出重要贡献的个人和集体，均已在文中作了明确说明并表示谢意。

作者签名：

日期：

基于多传感器的远程温度监测系统

摘要

本次毕业设计为实现一个远程温度采集系统，该系统基于物联网，可以扩展任意多个传感器。系统主要包括温度信息采集传送，后台服务器以及前端浏览器展示三大核心。

温度信息采集主要使用 ESP8266 WIFI 芯片通过 DHT 22 传感器采集环境温度信息，在指定的采集频率下将温度信息发往后台服务器。

后台服务器采用 Node.js 实现，为前端和温度信息采集部分提供 REST API 服务。服务器接收到温度采集部分传送过来的温度信息后，将温度信息储存到数据库中，供浏览器端随时查看温度信息。

前端浏览器展示利用 AngularJS 作为前端框架，Chart.js 图表库可视化温度数据，Bootstrap 作为样式库并提供响应式布局。通过 HTTP 协议获取后台的温度数据绘制温度曲线。支持实时温度曲线和历史温度查看。

关键词: ESP8266；物联网；Node.js；REST API; AngularJS；

Remote temperature monitoring system based on multi sensor

ABSTRACT

The graduation design is for coming true a remote temperature acquisition system, the system is based on Internet of things, can be extended to arbitrary multiple sensors.Information acquisition system mainly includes three cores:the temperature transmission,backstage server and front-end web browser.

Temperature information collection is mainly using ESP8266 wi-fi chips through DHT 22 temperature sensor to collect environmental information, temperature information is sent to the backend server under the specified acquisition frequency .

Backstage server using Node. Js to implement, provides the front part and temperature information acquisition with REST API service.Server receives the temperature information of the temperature acquisition part, the temperature information is stored in the database, for the browser to check the temperature information at any time.

Front-end web browser display use AngularJS as a front-end framework, Chart. Js charts library visual temperature data, the Bootstrap as style library and provide responsive layout.Get the background temperature curve drawing temperature data through the HTTP protocol.Support real-time temperature curve and the historical view.

Key words: ESP8266 ; IOT; Node.js ;REST API; AngularJS ;

目录

第一章 绪论

1.1 选题背景

随着科技的发展，传统靠人工控制的温度、湿度、液位等信号的测压、力控系统，外围电路比较复杂，测量精度较低，分辨力不高，需进行温度校准；且它们的体积较大、使用不够方便，更重要的是参数的设定需要有其它仪表的参与，外界设备多，成本高，因而越来越适应不了社会的要求。

而随着物联网这个话题越来越火。物联网开始成为新一代信息技术的重要组成部分。其英文名称是 The Internet of things。利用物联网可以广泛应用各种感知技术。大量的传感器可以部署在物联网中，每个传感器都能够从外界采集信息，不同类的传感器能够捕获的信息不同。而且可以按照一定规律实时采集，更新数据。

物联网与传统行业的融合是必然的发展趋势。智能家居、车联网等领域已经出现了一些领军的初创型企业。另外在医疗行业,根据调查,2012 年中国可穿戴移动医疗设备市场销售规模达到 4.2 亿元,预计到 2015 年这一市场规模将超过 10 亿元,2017 年将接近 50 亿元。不管是以家电为中心的智能家居还是以汽车为中心形成的车联网,通过物联网平台不仅可以实现远程的监控或者报警,而且与云平台结合进行扩展,提供各种服务。更多的传统企业将其产品与物联网结合,逐渐形成了以产品为中心延伸到以服务为中心。

1.2 设计内容

本次毕业设计为实现一个远程温度采集系统，该系统基于物联网，可以扩展任意多个传感器。系统主要包括温度信息采集传送，后台服务器以及前端浏览器展示三大核心。

温度信息采集主要使用 ESP8266 WIFI 芯片通过 DHT 22 传感器采集环境温度信息，在指定的采集频率下将温度信息发往后台服务器。

后台服务器采用 Node.js 实现，为前端和温度信息采集部分提供 REST API 服务。服务器接收到温度采集部分传送过来的温度信息后，将温度信息储存到数据库中，供浏览器端随时查看温度信息。

前端浏览器展示利用 AngularJS 作为前端框架，Chart.js 图表库可视化温度数据，Bootstrap 作为样式库并提供响应式布局。通过 HTTP 协议获取后台的温度数据绘制温度曲线。支持实时温度曲线和历史温度查看。

1.3 设计的目的和意义

本次设计探索了基本的物联网开发方式,尝试将最新的 HTML5 技术用于传统的数据采集系统中，改变传统数据采集系统界面不美观，操作复杂的缺点。最终用户不需要专用的设备查看结果，只要使用常用的智能手机，PC 或平板电脑的浏览器就能随时查看温度信息。

第二章 系统总体设计方案

本次毕业设计为实现一个远程温度采集系统，该系统基于物联网，可以扩展任意多个传感器。系统主要包括温度信息采集传送，后台服务器以及前端浏览器展示三大核心。系统框图如图 1-1：

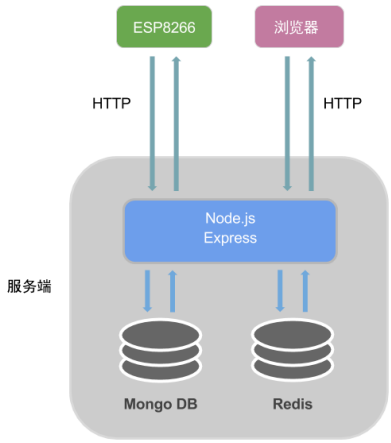

图 1-1 系统框图

温度信息采集部分主要使用 ESP8266 WIFI 芯片通过 DHT 22 传感器采集环境温度信息，在指定的采集频率下，利用 WIFI 通过 HTTP 协议将采集到的温度信息发往后台服务器。

后台服务器采用 Node.js 实现，使用的是 Node.js 下的 Express WEB 开发框架，为前端和温度信息采集部分提供 REST API 服务。服务器接收到温度采集部分传送过来的温度信息后，将温度信息储存到基于内存的 Redis 数据库，作为一个高性能实时的温度信息队列，供浏览器端随时查看实时温度曲线。在指定的周期将温度信息存储到 MongoDB 数据库中持久化存储温度数据，提供历史数据查看。

前端浏览器展示利用 AngularJS 作为前端框架，Chart.js 图表库可视化温度数据，Bootstrap 作为样式库并提供响应式布局。通过 HTTP 协议获取后台的温度数据绘制温度曲线。支持实时温度曲线和历史温度查看。

2.1 服务端技术方案

2.1.1 Node.js

服务端的编程语言使用的是 Node.js ，Node.js 采用 C++语言编写而成，是一个 Javascript 的运行环境。在最开始的时候，JavaScript 只能运行在浏览器中。而 Node.js 允许在后端（脱离浏览器环境）运行 JavaScript 代码。Node.js 采用了 Google Chrome 浏览器的 V8 javascript 解析引擎，性能非常好，同时还提供了很多系统级的 API，如文件操作、网络编程等。浏览器端的 Javascript 代码在运行时会受到各种安全性的限制，对客户系统的操作非常有限。相比之下，Node.js 则是一个全面的后台运行时，为 Javascript 提供了和其他语言一样能够实现的许多功能。

除此之外，伴随着 Node.js 的还有许多有用的程序模块，它们可以简化很多重复的劳作。因此，实际上 Node.js 既是一个运行时环境，同时又是一个巨大的代码库。

2.1.2 Express

Express 是目前非常流行的基于 Node.js 的 Web 开发框架，可以快速地搭建一个完整功能的 Web 应用，他对 nodejs 做了一定的封装，提供了一系列强大特性和丰富的 HTTP 工具。以方便建立自己的 web 应用，而不需要再使用 nodejs 原始的方法创建 http 服务。

2.1.3 MongoDB

MongoDB 是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。他支持的数据结构非常松散，是类似 json 的 bjson 格式，因此可以存储比较复杂的数据类型。MongoDB 最大的特点是他支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引。它的特点是高性能、易部署、易使用，存储数据非常方便。

2.1.4 Redis

Redis 是一个开源，先进的 key-value 内存数据库， Redis 数据库中的所有数据完全在内存中存储，由于内存的读写速度远远快于普通硬盘甚至机械硬盘，因此操作 Redis 数据库的速度非常快。相比许多键值数据存储，Redis 拥有一套较为丰富的数据类型。Redis 支持最大多数开发人员已经知道像列表，集合，有序集合，散列数据类型。这使得它非常容易解决各种各样的问题。Redis 还是一个多实用的工具，可以在多个用例如缓存，消息，队列使用(Redis 原生支持发布/订阅)，任何短暂的数据，应用程序，如 Web 应用程序会话，网页命中计数等。

2.1.5 RESTful API

API 全称叫 Application Programming Interfaces（应用程序编程接口）。用来提供访问应用程序的接口，其它想要访问此程序的设备或软件不需要知道这个 API 是怎么实现的，只要会调用这个接口就能调用程序的功能。

RESTful 是一种设计风格而不是一个标准。REST 通常基于使用 HTTP，URI，和 XML（标准通用标记语言下的一个子集）以及 HTML（标准通用标记语言下的一个应用）这些现有的广泛流行的协议和标准。REST 定义了一组体系架构原则，可以根据这些原则设计以系统资源为中心的 Web 服务，包括使用不同语言编写的客户端如何通过 HTTP 处理和传输资源状态。 如果考虑使用它的 Web 服务的数量，REST 近年来已经成为最主要的 Web 服务设计模式。

2.2 硬件方案

2.2.1 ESP8266

ESP8266 芯片是一个完整且自成体系的 Wi-Fi 网络解决方案，它能够搭载软件应用自己作为处理器，或作为另一个应用处理器模块来支持 Wi-Fi 功能。ESP8266 在自己搭载应用并作为设备中唯一的应用处理器时，能够从外接闪存中直接启动。内置的高速缓冲存储器有利于减少内存需求，提高系统性能。另外一种情况是，无线上网接入承担 Wi-Fi 适配器的任务时，可以将其添加到任何基于微控制器的设计中，连接简单易行，只需通过 SPI/SDIO 接口或中央处理器 AHB 桥接口即可。 ESP8266 强大的存储和片上处理能力，使其可通过

GPIO 口集成传感器及其他应用的特定设备，实现了最低前期的开发和运行中最少地占用系统资源。ESP8266 高度片内集成，包括天线开关 balun、电源管理转换器，因此仅需极少的外部电路，且包括前端模块在内的整个解决方案在设计时将所占 PCB 空间降到最低。

2.2.2 DH22

DH22 数字温湿度传感器是一款含有已校准数字信号输出的温湿度复合传感器。它应用专用的数字模块采集技术和温湿度传感技术，确保产品具有极高的稳定性与卓越的长期可靠性。传感器包括一个 NTC 测温元件和一个电容式感湿元件，并与一个高性能 8 位单片机相连接。因此该产品具有品质卓越、超快响、抗干扰能力强、性价比极高应等优点。每个 DH22 传感器都在极为精确的湿度校验室中进行校准。校准系数以程序的形式储存在 OTP 内存中，传感器内部在检测信号的处理过程中要调用这些校准系数。单线制串行接口，使系统集成变得简易快捷。超小的体积、极低的功耗，信号传输距离可达 20 米以上，使其成为各类应用甚至最为苛刻的应用场合的最佳选则。

2.3 前端技术方案

2.3.1 AngularJS

AngularJS 是 Google 推出的一款 Web 应用开发框架。它提供了一系列兼容性良好并且可扩展的服务，包括数据绑定、DOM 操作、MVC 设计模式和模块加载等。AngularJS 是为了克服 HTML 在构建应用上的不足而设计的。HTML 是一门很好的为静态文本展示设计的声明式语言，但要构建 WEB 应用的话它就显得乏力了。AngularJS 通过使用我们称为指令(directives)的结构，让浏览器能够识别新的语法。例如：使用双大括号{{}}语法进行数据绑定；使用 DOM 控制结构来实现迭代或者隐藏 DOM 片段；支持表单和表单的验证；能将逻辑代码关联到相关的 DOM 元素上；能将 HTML 分组成可重用的组件。

2.3.2 Bootstrap

Bootstrap 是由 Twitter 发布的用于快速开发 Web 应用程序和网站的开源前端样式和组件库。目前，它也是 GitHub 上 Star 最多的项目。Bootstrap 是

基于 HTML、CSS、JAVASCRIPT 的。包含了大量的样式库和功能强大的组件。利用其提供的栅格系统可以很容易快速地开发出适配多屏幕的 WEB 应用。

2.3.3 Chart.js

Chart.js 是一个基于 HTML5 canvas 技术的开源图表绘制工具库。Chart.js 简化了在网站上绘制动态图表的工作。Chart.js 帮用户用不同的方式使数据变得可视化。每种类型的图表都有动画效果，效果看上去非常棒。Chart.js 不依赖任何外部工具库，轻量级（压缩之后仅有 4.5k），并且提供了加载外部参数的方法。

第三章 后台服务器设计

服务器（Server）指：一个管理资源并为用户提供服务的计算机软件，通常分为文件服务器（能使用户在其它计算机访问文件），数据库服务器和应用程序服务器。运行以上软件的计算机，或称为网络主机（Host）。一般来说，服务器通过网络对外提供服务。可以通过 Intranet 对内网提供服务，也可以通过 Internet 对外提供服务。

本次设计中的后台服务器主要功能是接收 ESP8266 硬件端传送过来的温度信息，并将其储存到数据库中。当前端浏览器请求数据时为其提供温度数据信息。

3.1 HTTP 请求方法

HTTP 协议使用不同的请求方法（也叫“动作”）来以不同方式操作指定的资源，最常用的请求方法有以下四种：

GET：向指定的资源发出获取请求。使用 GET 方法应该只用在读取数据，而不应当被用于产生“副作用”的操作中。

POST：向指定资源提交数据，请求服务器进行处理（例如提交表单或者上传文件）。数据被包含在请求体中。这个请求可能会创建新的资源或修改现有资源，或二者皆有。

PUT：向指定资源位置上传其最新内容。

DELETE：请求服务器删除 Request-URI 所标识的资源。

方法名称是区分大小写的。当某个请求所针对的资源不支持对应的请求方法的时候，服务器应当返回状态码 405（Method Not Allowed），当服务器不认识或者不支持对应的请求方法的时候，应当返回状态码 501（Not Implemented）。

HTTP 服务器至少应该实现 GET 和 HEAD 方法，其他方法都是可选的。此外，除了上述方法，特定的 HTTP 服务器还能够扩展自定义的方法。

3.2 Express 框架

3.2.1 Express 示例

后台服务器框架主要是用 Node.js 下的 Express Web 框架实现。使用 Express 来搭建 WEB 应用非常简单，基本的 Express 应用如下：

```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send('Hello World!');
});

var server = app.listen(3000, function () {
  var host = server.address().address;
  var port = server.address().port;

  console.log('Example app listening at http://%s:%s', host,
port);
});
```

上面的代码启动一个服务并监听从 3000 端口进入的所有连接请求。他将对所有向(/) URL 发出的 GET 请求将会 返回 “Hello World!” 字符串。对于其他所有路径全部返回 404 Not Found。

HTTPie 是一个开源的命令行下的 HTTP 客户端。它可以很方便的用来测试 WEB 服务。运行刚刚的 Express 示例，使用 HTTPie 来测试刚刚启动的应用。测试结果如下:

```
> http get localhost:3000
  HTTP/1.1 200 OK
  Connection: keep-alive
  Content-Length: 12
  Content-Type: text/html; charset=utf-8
  Date: Wed, 04 May 2016 04:17:51 GMT
  ETag: W/"c-7Qdih1MuhjZehB6Sv8UNjA"
  X-Powered-By: Express

     Hello World!
```

可以看到 HTTPie 工具向服务端的指定路径发出 get 请求后服务器返回了 Hello World 字符串。

3.2.2 Express 路由

路由是指如何定义应用的端点（URIs）以及如何响应客户端的请求。

路由是由一个 URI、HTTP 请求（GET、POST 等）和若干个句柄组成，Express 路由的结构如下： app.METHOD(path, [callback...], callback)，app 是 express 对象的一个实例， METHOD 是一个 HTTP 请求方法， path 是服务器上的路径， callback 是当路由匹配时要执行的函数。

下面是一个基本的路由示例：

```
var express = require('express');
var app = express();
// 匹配到GET请求时返回Hello world
app.get('/', function(req, res) {
  res.send('hello world');
});
```

示例中程序监听到客户端向指定的路径以 GET 的方式发送请求后，服务端会向客户端返回 Hello world。

3.2.3 Express 路由方法

路由方法源于 HTTP 请求方法，和 express 实例相关联。下面这个例子展示了为应用跟路径定义的 GET 和 POST 请求：

```
// GET 方法的 路由
app.get('/', function (req, res) {
  res.send('GET request to the homepage');
});

// POST 方法的路由
app.post('/', function (req, res) {
  res.send('POST request to the homepage');
});
```

当客户端向服务器指定路径发出 GET 请求后，响应的是对应的 GET 方法的回调函数，当客户端向服务器指定发出 POST 请求后，响应的是 POST 回调函数。

Express 定义了如下和 HTTP 请求对应的路由方法： get, post, put, head, delete, options, trace, copy, lock, mkcol, move, purge,

propfind, proppatch, unlock, report, mkactivity, checkout, merge, m-search, notify, subscribe, unsubscribe, patch, search, 和 connect。最常用的 HTTP 请求方法是 get, post, put, delete 这四种。

3.3 MongoDB 数据库

在本次设计中，MongoDB 做为系统的数据库，用来存储设备传感器信息以及对应传感器的温度信息。

3.3.1 mongoose

mongoose 是在 Node.js 环境中操作 MongoDB 数据库的一种便捷的封装，一种对象模型工具，类似 ORM，它提供了大量的方法去操作 MongoDB 数据库。将数据库中的数据转换为 JavaScript 对象以供用户在应用中使用或者将程序中的对象储存到 MongoDB 数据库中。

3.3.2 数据库表结构设计

本次设计中服务器端需要存储到数据库中的信息有设备信息以及传感器采集到的温度信息。需要设计数据库表来存储对应的信息。

要设计的设备信息表结构是这样的：

表名	数据类型	作用
device_id	Number	设备 ID
name	String	设备名字
profile	String	设备的备注信息
creat_at	Number	时间戳，记录设备创建的时间
sensors	文档类型	设备包含的传感器

其中，传感器信息表结构如下：

表名	数据类型	作用
sensor_id	Number	传感器的 ID
name	String	设备名字
sensor_type	String	传感器类型

使用 Mongoose 根据上表设计设备信息的数据模型代码如下：

```
var mongoose  = require('mongoose');
var Schema    = mongoose.Schema;

var SensorSchema = new Schema({
    sensor_id: { type: Number },
    name: { type: String,default:'temperature sensor'},
    sensor_type: { type: String }
});

var DeviceSchema=new Schema({
    device_id:{type:Number,index:true,unique:true},
    name:{type:String},
    profile:{type: String,default:''},
    create_at: { type: Number, default: new Date().getTime() },
    sensors:[SensorSchema]
});
mongoose.model('Device', DeviceSchema);
```

实际上存储到 MongoDB 中的数据结构是这样的：

```
{
    "_id" : ObjectId("56e576b433dd1b800491d5bd"),
    "device_id" : 13867330,
    "name" : "教室的设备",
    "sensors" : [
        {
            "sensor_id" : 371394,
            "sensor_type" : "温度传感器",
            "_id" : ObjectId("56e576b433dd1b800491d5bf"),
            "name" : "温度传感器"
        }
    ],
    "create_at" : NumberLong(1457878708507),
    "profile" : "测量教室里的温度",
    "__v" : 0
}
```

可以看到，传感器信息表是设备表的子文档，存储在设备表的 Sensors 里面。这样，当向数据库中查询设备信息的时候，同时也获取到了设备包含的所有的传感器信息。而对于普通的非关系型数据库，是不能嵌套的，想要表达这种嵌套的关系只能再新建一个表，表之间通过外键关联。如果想要同时查询设备和设备包含的传感器，需要先找到设备，然后再根据设备的 ID 找到传感器。这样不便于理解也不便于编程。相对于普通的关系数据库，MongoDB 存储的数据可以嵌套，就像 Json 格式一样，这样非常便于理解。查询数据的时候也特别方便。

另外还需要设计一张表用来存储传感器采集到的温度信息。温度信息表结构如下：

表名	数据类型	作用
sensor_id	Number	传感器的 ID
value	Number	传感器采集的数据值
timestamp	Number	时间戳，数据采集的时间

使用 Mongoose 根据上表设计设备信息程序如下：

```
var mongoose   = require('mongoose');
var Schema     = mongoose.Schema;

var SensorValueSchema = new Schema({
    sensor_id: { type: Number },
    value: { type: Number},
    timestamp:{type: Number, default: new Date().getTime()}
});

mongoose.model('SensorValue', SensorValueSchema);
```

3.4 Redis 数据库

系统需要查看实时的温度曲线，但是又不需要全部存储到数据库中，十分钟甚至一小时存储一次温度信息都是可以接受的。假如每个传感器以每秒两次

的频率向服务器发送数据。每个设备可以有多个传感器，系统又会有多个设备。这样当设备越来越多，系统需要不断地向 MongoDB 数据库中写入数据，而 MongoDB 数据是存储在硬盘中的，相对于内存来说，访问硬盘的代价是很大的，速度非常慢。而 Redis 是基于内存的数据库，所有数据都存储在内存中，访问速度很快。因此选用 Redis 作为缓存队列提供实时数据存储。

3.4.1 Redis 数据类型

Redis 支持 5 种类型的数据类型：

字符串：键值对，每个键对应一个字符串值

哈希：键值对的集合。Redis 的哈希值是字符串字段和字符串值之间的映射，因此它们被用来表示对象。

列表：Redis 的列表是简单的字符串列表，排序插入顺序。可以添加元素到 Redis 的列表的头部或尾部。

集合：Redis 的集合是字符串的无序集合。在 Redis 您可以添加，删除和测试文件是否存在，在成员 O（1）的时间复杂度。

有序集合：Redis 的有序集合类似于 Redis 的集合，字符串不重复的集合。不同的是，每个成员都可以有一个评分，通过评分可以用来排序或者查询想要的数据。

3.4.2 Redis 实时队列系统

本次使用 Redis 的目的是作为一个高性能队列系统，记录每个传感器最新十分钟的采样数据。根据分析，最适合使用的 Redis 数据类型是有序集合来设计队列系统。其中，集合的成员就是传感器的采样数据，使用时间戳（最小单位微妙）作为成员的评分，这样可以根据数据的采样时间来排序或删除过期数据。因为有序集合的成员是唯一且不能重复的，但是传感器不同时间采样的数据值是可以相同的，因此，在传感器的数据值前拼接上当前的时间戳作为成员，这样每个成员就是唯一的了。

向 Redis 中写入数据的代码如下：

```
  redisClient.zadd(sensor_id, timestamp, (timestamp * 10000) + value,
    function(err, response) {
       if (err) throw err;
       console.log('added ' + response + ' items.');
     });

  redisClient.zremrangebyscore(sensor_id, 0, timestamp - tenMin,
    function(err, response) {
       if (err) throw err;
       console.log('removeAll ' + response + ' items.');
     });
```

代码主要作用就是先把时间戳拼接到数据上防止数据重复。然后向以传感器的 ID 作为 key 的有序集合中插入拼接好的的数据点作为成员，当前时间戳作为评分。并且通过评分来移除过期的数据。

从 Redis 中获取数据的代码如下：

```
  //移除十分钟之前的数据
  redisClient.zremrangebyscore(sensor_id, 0, timestamp - tenMin,
function(err, response) {
    if (err) throw err;
    console.log('removeAll ' + response + ' items.');
    //获取最新的数据
    redisClient.zrangebyscore(sensor_id, latsTimestamp + 1, forever,
function(err, response) {
      if (err) throw err;

      var timestamp = parseInt(_.last(response) / 10000);
      _.each(response, function(data, index) {
        response[index] = response[index] % 10000;
      });

      res.json({
        data: response,
        latsTimestamp: timestamp
      });

    });
  });
```

代码功能首先移除过期的数据，然后根据时间戳获取对应范围的传感器数据，并移除数据前面拼接的时间戳信息，就是实时的传感器数据了。

这样，基于 Redis 的高性能队列系统就设计完成了。

3.5 设计 REST API

系统的主要功能功能已经明确。现在需要向外界暴露接口供其它应用调用了。API 用来对外界提供服务，在本次设计中，API 对硬件提供信息存储服务，对浏览器提供数据获取服务。

根据设计目的，系统应该有设备信息相关 API 和传感器信息相关 API.

3.5.1 设备信息 API

设备信息 API 主要是提供设备信息的获取，创建，修改，删除等操作，对应着 HTTP 方法是 GET,POST,PUT,DELETE。

对于设备 API 设计的 URL 地址是 ：

```
api/v1.0/devices
```

通过以上地址可以操作设备信息，对应 URL 的 GET 请求回调函数如下：

```
exports.get = function (req, res, next) {
  Device.find(function(err,devices){
    res.json(devices);
  });
};
```

当匹配到路径的 GET 请求后，从数据库中取出所有的设备信息并且以 Json 的格式返回。

对应 URL 的 Post 请求回调函数如下：

```
exports.creat = function (req, res, next) {
  Device.create(req.body.device,function(err,sensors){
    res.json(sensors);
  });
};
```

当匹配到 POST 路径后，也就是需要创建一个设备，服务器程序从请求体中取出需要创建的设备信息，然后将其写入 MongoDB 数据库中。

3.5.2 创建温度信息 API

温度信息 API 提供温度信息的创建。

对于温度信息的创建 API，设计的 URL 地址是：

```
api/v1.0/sensor/:sensor_id/value
```

创建温度信息 API 只响应 POST 方法，回调函数如下：

```
exports.post = function(req, res, next) {
  var sensor_id = Number(req.params.sensor_id);
  var value = req.body.value;

  var timestamp = new Date().getTime();

  redisClient.zadd(sensor_id, timestamp, (timestamp * 10000) +
value, function(err, response) {
    if (err) throw err;
    console.log('added ' + response + ' items.');
  });

  redisClient.zremrangebyscore(sensor_id, 0, timestamp - tenMin,
function(err, response) {
    if (err) throw err;
    console.log('removeAll ' + response + ' items.');
  });

  if (req.body.save) {
    SensorValue.create({
      sensor_id: sensor_id,
      value: value
    }, function(err, sensors) {
      if (err) {
        return false;
      }
      res.json(sensors);
    });
  } else {
    res.send({
      success: true
    });
  }
};
```

当服务器匹配到对创建温度信息 URL 的 POST 请求后，函数会将 POST 过来的数据放入到 Redis 队列中。并且在规定的周期下将数据存入到 MongoDB 中持久化数据作为历史记录。

3.5.3 历史数据记录 API

历史数据记录 API 提供历史数据的查询。历史数据记录 API URL 如下：

```
api/v1.0/sensor/:sensor_id/value/history
```

历史记录 API 只响应 GET 请求方式，回调函数如下：

```
  exports.getHistory = function(req, res, next) {
  var sensor_id = req.params.sensor_id;
  var start = req.query.start || 0;
  var end = req.query.end || 0;

  SensorValue.find({
      sensor_id: sensor_id
    })
    .select('value timestamp')
    .where('timestamp').gte(start).lte(end)
    .sort('timestamp')
    .exec(function(err, sensorValues) {
      if (err) {
        return false;
      }
      res.json(sensorValues);
    });
};
```

当服务器匹配到对历史记录 URL 的 GET 请求后，先从请求路径中获取传感器的 ID,起始时间戳和结束时间戳，然后根据传感器 ID,和时间戳查询 MongoDB 数据库，返回查询到的结果。

3.5.3 实时温度 API

实时温度 API 提供实时温度显示的功能。实时温度 API URL 路径如下：

```
api/v1.0/sensor/:sensor_id/value/realtime
```

实时温度 API 响应 GET 请求，回调函数如下：

```
exports.realTime = function(req, res, next) {
  var sensor_id = Number(req.params.sensor_id);//获取传感器ID
  var latsTimestamp = Number(req.query.latsTimestamp) || 0;//上次
获取数据的时间戳
  var timestamp = new Date().getTime();//当前时间戳
  //移除十分钟之前的数据
  redisClient.zremrangebyscore(sensor_id, 0, timestamp - tenMin,
function(err, response) {
    if (err) throw err;
    console.log('removeAll ' + response + ' items.');
    //获取最新的数据
    redisClient.zrangebyscore(sensor_id, latsTimestamp + 1,
forever, function(err, response) {
      if (err) throw err;

      var timestamp = parseInt(_.last(response) / 10000);
      _.each(response, function(data, index) {
        response[index] = response[index] % 10000;
      });
      res.json({
        data: response,
        latsTimestamp: timestamp
      });
    });
  });
};
```

函数主要功能就是从请求路径中获取传感器 ID 以及上次获取数据的时间戳，然后把时间戳大于上次的数据返回。这样就可以获取最新的温度信息了。当没有最新的温度信息时返回为空。

第四章 硬件设计

4.1 NodeMCU 开发板

NodeMCU 是一款基于 ESP8266 WIFI 芯片的开源硬件原型平台，包括固件和开发板，用几行简单的 Lua 脚本就能开发物联网应用。

因为 NodeMCU 开发板设计紧凑美观，而且带有 USB 转串口芯片，并且引出了 GPIO 口，可以方便的进行 ESP8266 开发。在本次设计中使用的只是 NodeMCU 的开发板,而并没有使用 NodeMCU 的固件开发。Node 硬件开发板如图 4-1：

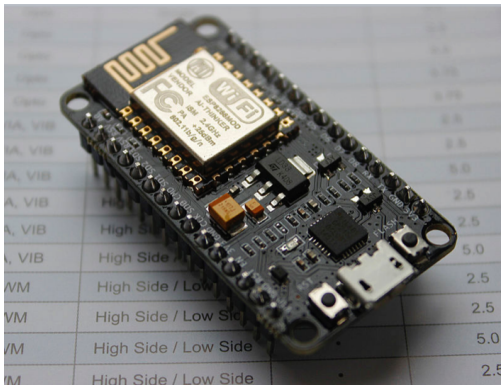

图 4-1：NODEMCU 开发板

NodeMCU 对应的引脚图如图 4-2：

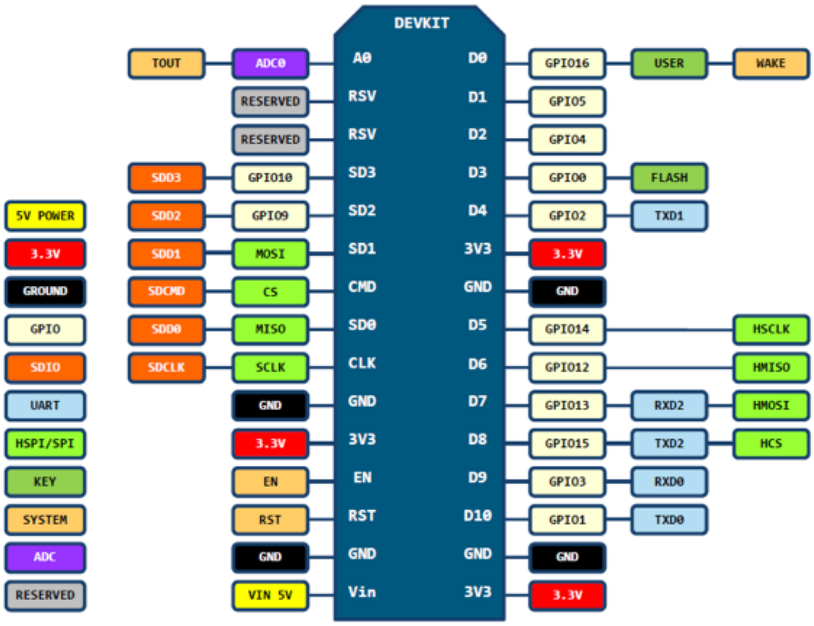

图 4-2：NODEMCU 引脚图

4.2 Sming 库

Sming 是 Github 上的一个 ESP8266 芯片的开源高性能编程框架，基于 C++ 语言。本次设计中硬件的编程框架选用的就是 Sming。因为其具有以下特性：

快速，用户友好的开发方式。

像 Arduino 一样操作 GPIO。

高效的性能和内存占用。

符合 Arduino 库函数标准，几行代码就可以使用任何流行的外设。

内建 spiffs 文件系统。

内建高性能网络和无线模块。

内建 JSON 操作库。

支持 HTTP,AJAX,Websockets。

网络基于 LWIP 协议栈。

Sming 库具有的这些特性大大简化了硬件开发过程。使得用户可以专注于创意的实现而不用苦于如何实现网络协议，如何编写传感器驱动，因为这些库都帮我们做好了。例如，读取 DHT22 温湿度传感器数据，使用 Sming 库只要以下代码;

```
#include <Libraries/DHT/DHT.h>

#define WORK_PIN 0 // GPIO0
DHT dht(WORK_PIN, DHT22);

void init()
{
  dht.begin();

  float h = dht.readHumidity();//读取湿度信息
  float t = dht.readTemperature();//读取温度信息
}
```

可以看到，代码非常简单明了，使用面向对象的思想操作硬件。如果以传统的方式开发，读取一个 DH22 传感器的温湿度信息。需要配置 IO 口的模式，编写单总线协议的驱动，查看 DH22 数据手册发送指定的命令才能完成。所以，库开发的好处还是显而易见的。

4.3 硬件主程序

硬件使用 Sming 库开发，主要是向服务器的温度信息 API 以 POST 的方式发送采集到的温度信息。系统初始化程序如下：

```
   // 程序初始化函数
  void init() {
    system_set_os_print(0);
    ledInit();
    ledON();
    Serial.begin(SERIAL_BAUD_RATE);  // 默认波特率115200
    // 连接到 WIFI
    WifiStation.config(WIFI_SSID, WIFI_PWD);
    WifiStation.enable(true);
    WifiAccessPoint.enable(false);

    //成功连接WIFI连接回调函数
    WifiStation.waitConnection(
        connectOk, 20,
        connectFail);

  }
```

初始化程序主要功能就是配置串口，连接 WIFI,当连接 wifi 成功后调用 connectOk 回调函数，回调函数如下：

```
String postURL = "http://192.168.1.192:3000/api/v1.0/sensor/";
  // Will be called when WiFi station is connected to AP
void connectOk() {
  Serial.println("连接成功 :)");
  Serial.println("设备IP地址为：");
  Serial.println(WifiStation.getIP());
  ShowInfo();
  postURL = postURL + String(system_get_chip_id()) + "/value";
  checkTimer.initializeMs(500, postDate).start();
}
```

回调函数的功能主要是通过串口打印系统相关信息，并且根据系统芯片的 ID 生成 POST 的路径，然后启动一个定时器以 500ms 的周期调用发送温度信息函数向服务器发送温度信息。发送温度信息的函数如下：

```
void postDate() {
  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature();
  ledToggle();
  httpClient.reset(); // Clear any data from previous requests.
  httpClient.setRequestContentType("application/json");

   String postString ="{\"value\":" + String(temperature)
  +",\"save\":" + getSaveValue() + "}";

  httpClient.setPostBody(postString);
  httpClient.doPost(postURL, onDataReceived);
}
```

发送函数的主要功能是读取传感器的温度信息，将其拼接成 JSON 格式的字符串，然后通过 postURL 发送到服务器。至此，硬件端的编程就完成了。

第五章 前端浏览器程序设计

5.1 Angular.js 单页应用

Angular 是谷歌开源的一款前端开发框架，利用 Angular.js 可以很方便的构建出单页应用（SPAs：Single Page Applications）。

单页应用并不是说网页只有一个页面。单页应用是指在浏览器中运行的应用，它们在使用期间不会重新加载页面。而传统的网页，每次点击一个连接之后，网页都需要重新刷新，然后加载并跳转到新页面。单页应用具有以下优点：

1、具有桌面应用的即时性、网站的可移植性和可访问性。

2、用户体验好、快，内容的改变不需要重新加载整个页面，web 应用更具响应性和更令人着迷。

3、基于上面一点，SPA 相对对服务器压力小。

4、良好的前后端分离。SPA 和 RESTful 架构一起使用，后端不再负责模板渲染、输出页面工作只需要提供 API 就可以了。

5.2 Bootstrap 响应式布局

使用 Bootstrap 可以很方便的设计响应式布局应用。

响应式布局是 Ethan Marcotte 提出的一个网页设计概念，简而言之，就是一个网站一套代码能够兼容多个不同屏幕尺寸的终端——而不是为每个尺寸的屏幕终端做一个特定的版本。这个概念是为解决移动互联网浏览而诞生的。

响应式布局可以为不同屏幕尺寸终端的用户提供更加舒适美观的界面和更好的用户体验，而且随着目前大屏幕移动设备的普及，用大势所趋来形容也不为过。

本次设计采用了响应式布局，使得界面在小屏幕的智能手机，中等屏幕的平板电脑以及大屏幕的电脑显示器上都能有比较好的显示效果。系统在小屏幕上显示不会过于紧密导致无法观看使用，在大屏幕上也不会过于松散而浪费显示面积。以本系统的所有设备显示界面为例，当在大屏幕（通常是电脑显示器）上显示时，界面如图 5-1。

图 5-1 :大屏幕布局

而在中等屏幕（通常是平板电脑）上显示时，界面如图 5-2 左图。在小屏幕（通常是智能手机）上显示时，界面如图 5-2 右图。

图 5-2 :中等屏幕布局（左）和小屏幕布局（右）

可以看到同一个网站页面，在不同的屏幕上，系统界面会有不同的布局，这样增强了用户体验。

5.3 ng-Resource

AngularJS 提供了$resource Service 来更方便地与 RESTful 服务器 API 进行交互. $resource 可以方便地定义一个 REST 资源，而不必手动所有的声明 CRUD 方法。在本次设计中前端主要通过 ng-resource 来获取服务器端的数据。

以下代码创建一个 resource 服务，本次设计中所有与服务器交互都是通过此 resource 服务进行的。

```
(function() {
  'use strict';
  angular
    .module('myIot')
    .factory('resource',resource);
  /** @ngInject */
  function resource($resource) {
    var baseURL='http://192.168.1.192:3000/api/v1.0/';
    return {
      device:$resource(baseURL+'devices'),
      sensorValue:{ history:$resource(baseURL+
          'sensor/:sensor_id/value/history', { sensor_id:
          '@sensor_id' }),
      realtime:$resource(baseURL+'sensor/:sensor_id/value/
                realtime', { sensor_id: '@sensor_id' })
      }
    };
  }

})();
```

服务中通过$resource 创建了设备获取，历史记录获取和实时数据获取方法。其它想要获取数据的地方只要调用这个服务的相应方法就能获取到数据。

5.3 所有设备页面

所有设备页面需要获取所有的设备信息并将其展示出来。

只要在所有信息页面的 Controller 中注入上节中写的 resource 服务并调用 resource 服务的设备信息获取方法就能获取到所有设备信息。所有设备信息页面的 Controller 代码如下：

```
(function() {
  'use strict';

  angular
    .module('myIot')
    .controller('MyDeviceController', MyDeviceController);

  /** @ngInject */
  function MyDeviceController($state, resource, $log, webDevTec) {
    $log.debug('MyDeviceController init');
    var vm = this;

    resource.device.query(function(data) {
      vm.devices = data;
    });

    vm.getTheme=getTheme;
    function getTheme(index){
      var themes=['primary','success','warning','danger','info'];
      return themes[index%5];
    }

    vm.goDashboard = goDashboard;
    function goDashboard(device,sensor) {
      $state.go('dashboard', {
        device:device,
        sensor: sensor
      });
    }

  }
})();
```

所有设备页面 HTML 代码如下：

```
 <div class="row my-device">
  <div class="col-md-4 col-sm-6" ng-repeat="device in main.devices">
    <div class="panel panel-{{main.getTheme($index)}} ">
      <div class="panel-heading">
          <span class="device-name"><i class="fa fa-cog fa-
spin"></i> {{device.name}}</span>
      </div>
      <div class="panel-body">
        <div class="device-profile">
          <p>
            备注：{{device.profile}}<br>
          </p>
        </div>
```

```
        <div ng-repeat="sensor in device.sensors">
          <img class="logo" src="../../assets/images/temperature.png"
alt=""/>
            <div class="col-xs-8">
              设备类型: {{sensor.sensor_type}}<br>
              设备 ID: {{sensor.sensor_id}}
            </div>
            <input type="button" class="btn btn-default pull-right"
ng-click="main.goDashboard(device,sensor)" name="name" value="查看详情
">
        </div>
      </div>
    </div>
  </div>
</div>
```

所有设备页面的显示在大屏幕下显示效果如下：

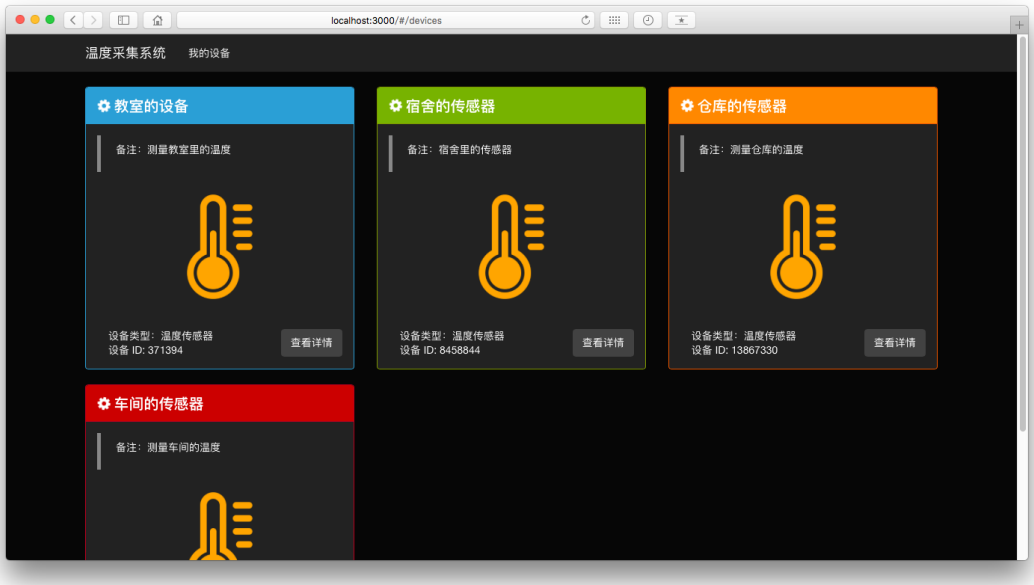

图 5-3 所有设备页面

5.4 传感器数据展示页面

当在设备列表点击查看详情按钮，会进入传感器的数据展示页面。数据展示页面也是通过前面的 resource 服务获取数据的。数据展示页面主要是两个图标的显示，一个是实时数据显示，另一个图标是历史日期记录显示。

实时数据显示的代码如下：

```
    var vm = this;
    //realtime chart start
    vm.realTimeChart = {
      labels: [],
      data: [
        []
      ],
      option: {
        animation: false,
        datasetFill: true,
        scaleOverride: true, //是否用硬编码重写y轴网格线
        scaleSteps: 4, //y轴刻度的个数
        scaleStepWidth: 10, //y轴每个刻度的宽度
        scaleStartValue: 0, //y轴的起始值
        responsive: true,
        scaleLabel: "<%=value%> ℃"
      }
    };

    var latsTimestamp = 0;
    var realTimer=$interval(function () {
      resource.sensorValue.realtime.get({
        sensor_id: sensor_id,
        latsTimestamp: latsTimestamp
      }, function (data) {
        if (data.data.length) {
          latsTimestamp = data.latsTimestamp;
          _.forEach(data.data, function (data) {
            vm.realTimeChart.labels.push('');
            vm.realTimeChart.data[0].push(data);
          });

          if (vm.realTimeChart.labels.length > 60) {
            vm.realTimeChart.labels.splice(0,
vm.realTimeChart.labels.length - 60);
            vm.realTimeChart.data[0].splice(0,
vm.realTimeChart.data[0].length - 60);
          }
        }
      });
    }, 500);

    $scope.$on('$stateChangeStart',  function(){
      $interval.cancel(realTimer);//页面切换后停止获取实时数据
    });
```

以上代码主要配置了实时数据显示的 Char.js 图表，然后以 500ms 的周期向服务器请求最新数据并刷新图表。

历史日期显示图表的相关主要代码如下：

```
vm.chart = {
      series: ['22'],
      option: {
        pointDot: true,
        scaleOverride: true, //是否用硬编码重写y轴网格线
        scaleSteps: 6, //y轴刻度的个数
        scaleStepWidth: 10, //y轴每个刻度的宽度
        scaleStartValue: -15, //y轴的起始值
        responsive: true,
        scaleLabel: "<%=value%> ℃",
        tooltipTemplate: "<%= value %>℃"
      }
    };
    //history chart start
    vm.histryChart = {
      labels: [],
      data: []
    };
    vm.historyDay = (new Date().getTime()) - 90 * 24 * 3600 * 1000;
    vm.showHistory = function () {
      resource.sensorValue.history.query({
        sensor_id: sensor_id,
        start: vm.historyDay + 1,
        end: vm.historyDay + 86400000
      }, function (data) {
        vm.histryChart.labels = [];
        vm.histryChart.data[0] = [];
        _.forEach(data, function (obj, index) {
          vm.histryChart.labels.push(index + '时');
          vm.histryChart.data[0].push(obj.value.toFixed(1));
        });
        //vm.histryChart.labels.splice(0,24);
      });
    };
    vm.showHistory();
    //history chart end
  }
})();
```

以上代码主要是配置历史日期图表，根据用户选择的日期向服务器请求温度数据并在获取到数据后刷新图表。

最终传感器的信息展示页面在大屏幕下的显示效果如下（历史数据记录为插入的随机数据）：

图 5-4 传感器信息展示页面

致谢

历时将近 4 个月的时间终于完成了这次的毕业设计，在毕设的设计中和论文的写作过程中遇到了无数的困难和障碍，都在同学和老师的帮助下度过了。感谢我的毕业设计指导老师周瑾老师，感谢他对我的指导和信任。让我顺利完成这次毕业设计。

感谢这篇论文所涉及到的各位学者。本文参考了数位学者的研究文献，如果没有各位学者的研究成果的帮助和启发，我将很难完成本篇论文的写作。

感谢毕设中使用的那些开源代码库的贡献组织和个人，没有你们的无私奉献，本次毕设基本不可能进行下去。

感谢我现在的实习公司和同事们，在我毕设设计的过程中帮助我解决了许多难题，让我受益匪浅。

由于我的学术水平有限，所写论文难免有不足之处，恳请各位老师和学友批评和指正！

参考文献

[1]郭远威.大数据存储 MongoDB 实战指南[M].北京:人民邮电出版社,2015.

[2]李子骅.Redis 入门指南[M].北京: 人民邮电出版社,2013.

[3]Guillermo Rauch.了不起的 Node.js:将 Javascript 进行到底[M].北京:电子工业出版社，2014.

[4]陶国荣.Angular JS 实战[M].北京: 机械工业出版社,2015.

[5]张志勇.现代传感器原理及应用 [M].北京: 电子工业出版社,2014.

[6] 刘伟荣，何云.物联网与无线传感器网络[M].北京: 电子工业出版社,2013.

附录