北京邮电大学计算机学院

网络编程技术

期末作业报告

陈子迪 2014211521 胡旭强 2014211516 袁振宇 2014211529

2017-1-14

需求分析

基本要求：

· 有基本的用户管理功能

· 用户可以添加删除需要监测管理的设备

· 用户可以查看设备列表及设备的当前状态

· 模拟物联网设备向服务器周期性上报状态

扩展要求（选做其中的几项）

· 可以查看设备的历史数据，如显示最近一段时间内的温度变化情况，灯的开关情况

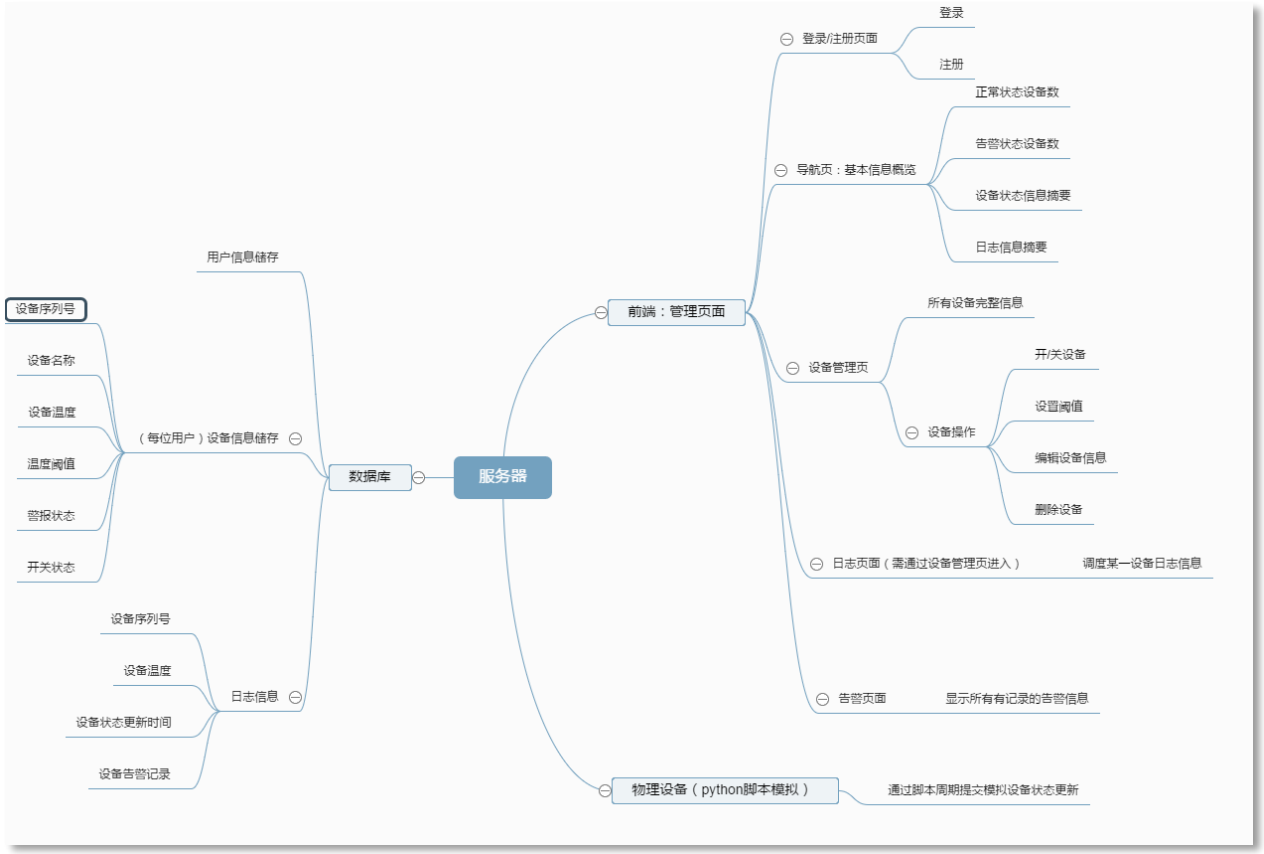
· 前端界面能及时显示设备状态变更情况

· 增加对设备的控制功能，如能向模拟的设备发送开关命令

· 增加告警功能，可以设置告警阈值，生成告警记录或者能向指定邮箱发送邮件

总体设计

根据项目（作业）要求，我们的总体设计如下。



根据图片，可以分为以下模块进行具体设计。

一、前端

1、登录/注册页面

在数据库中搭建用户信息数据库，之后通过服务器与前端进行交互来实现登录与注册。

2、主页

可采用 bootstrap 的 dash 模板进行修改，主页显示的信息包含了平台的基本信息。

处于正常、异常状态的设备数量统计，用直接数字显示；日志数量统计，直接数量显示；日志摘要，摘要出几条最新的日志信息并打上时间戳；设备状态摘要，摘要几台设备的一些信息；处于各温度段的设备数量的统计，采用图表的形式。

3、设备管理页面（实时刷新）

显示所有设备的完整信息，如设备名称、序列号、状态、温度、温度阈值、是否响起警报等；同时包含对设备的操作，包括添加、删除、编辑设备、开关设备。

在这一页进行实时动态刷新，能够通过表格反映出设备的状态变化。

在这一页添加历史纪录查看接口，进入每一个设备的历史纪录页。

4、历史记录页

在这一页显示出相应设备的历史纪录。

5、告警记录页

在这里生成过去时间内产生的告警记录报告，查看所有告警记录信息。

二、服务器

1、能够处理来自前端的请求，包括页面跳转等。

2、实现前端对数据库的访问和修改操作。

3、能够处理来自模拟设备的上报状态操作。

三、数据库

1、用户信息的记录

包括用户名、密码信息。

2、设备信息的记录

包括设备名称、序列号、状态、温度、温度阈值、是否响起警报等信息。

3、日志信息

包括日志所属用户、序列号、更新日期、时刻温度、警报状态等。

模块实现

一、前端

1、登录/注册页面

登陆模块接收前端发来的登陆用户名和密码，在数据库中查找比对，如比对成功，则进入系统管理界面，否则要求重新更新登陆。

登陆成功后，全局量会置为当前用户，而后各处理模块只能访问当前用户的信息，实现多用户数据的隔离。

注册：注册模块根据前端发来的用户名和密码，在数据库中增加一个新用户，一个空白的新用户。而后，可以用该用户名密码登录，进入管理界面，添加设备。

2、主页

①处于正常、异常状态的设备数量统计，用直接数字显示；日志数量统计，直接数量显示

直接访问数据库，查询设备总数得到总体数量，条件查找正常、报警状态设备数量可以得出两者数量。

②日志摘要，摘要出几条最新的日志信息并打上时间戳

从数据库中请求读取日志信息，根据时间降序排列，得到最新的几条日志记录返给前端。

③设备状态摘要，摘要几台设备的一些信息

从数据库中请求读取设备信息，读取几台设备的信息显示在表格中。

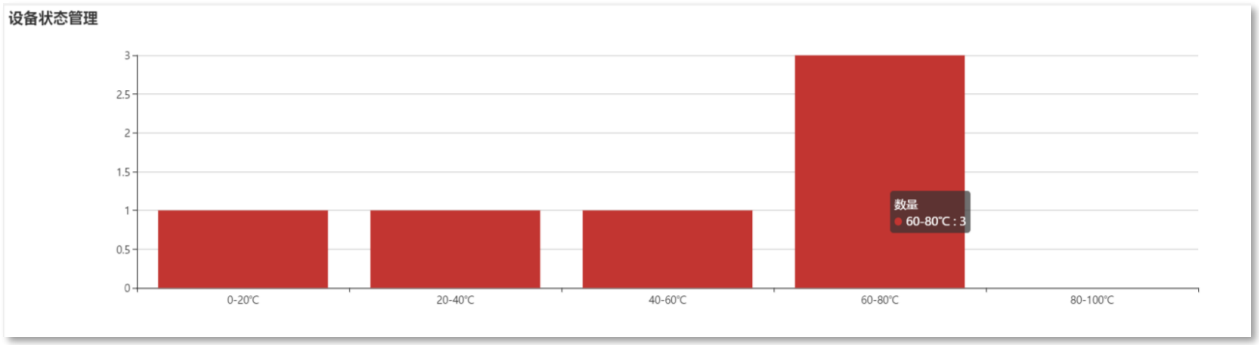
④处于各温度段的设备数量的统计，采用图表的形式

调用 ECharts 的 API，数据库向前端发送包含分类信息的 json 串，前端解析 json 串并用柱形图的形式将所处各温度段的设备数量显示出来。

```
var myChart = echarts.init(document.getElementById('main'));

  // 指定图表的配置项和数据
  function getDevTemNum()
  {

     myChart.setOption({
            title: {
               text: '设备状态管理'
            },
            tooltip: {},
            legend: {
               data:['0-20℃','20-40℃','40-60℃','60-80℃','80-100℃']
            },
            xAxis: {
               data: []
            },
            yAxis: {},
            series: [{
               name: '数量',
               type: 'bar',
               data: []
            }]
        });
}
```

```
function loadData()
{
   $.ajax({
      url:"/getDevTemNum",
      type:"GET",

      dataType:"JSON",
      success: function (data) {
         myChart.setOption({
            xAxis: {
               data: data.catagories
            },
            series: [{
               // 根据名字对应到相应的系列
               name: '数量',
               data: data.data
            }]
         });
      },
      error:function (jqXHR,textStatus,errorThrown) {
      }
   })
}
window.onload=loadData();
```

效果如下：



3、设备管理页面（实时刷新）

①显示所有设备的完整信息，如设备名称、序列号、状态、温度、温度阈值、是否响起警报等

从数据库中请求查询所有属于本用户的信息发送给前端，通过表格显示出来。通过内联 js，设计 showListFunc()函数来调用显示。

②同时包含对设备的操作，包括添加、删除、编辑设备、开关设备

用户可以添加删除需要监测管理的设备。使用 JSON 这种轻量级的文本数据交换格式，利用 AJAX 通过在后台与服务器进行少量数据交换，使网页实现异步更新，在不重新加载整个页面的情况下，与服务器交换数据并更新部分网页内容，实现设备的添加、修改和删除。

以下是添加编辑框的实现。编辑、删除操作类似。

```
$(document).ready(
    $("#btnAdd").click(
        function () {
            save_method = 'add';
            $('#form')[0].reset(); // 重置form

            $('#modal_form').modal('show'); // 显示modal
```

```
            $('.modal-title').text('添加设备'); // 设置 title
        }
    },
    $("#btnSave").click(
        function () {
            var url;
            if (save_method == 'add') {
                url = "/api/add/";
            }
            else {
                url = "/api/edit/";
            }
            $.ajax({
                url: url,
                type: "POST",
                data: $('#form').serialize(),
                dataType: "JSON",
                success: function (data) {
                    //如果成功，隐藏弹出框并重新加载数据
                    $('#modal_form').modal('hide');
                    showListFunc();
                },
                error: function (jqXHR, textStatus, errorThrown) {
                    alert('新建或添加错误！');
                }
            })
        }
    ),

.....

$(document).ready(
    $("#btnAdd").click(
        function () {
            save_method = 'add';
            $('#form')[0].reset(); // 重置 form

            $('#modal_form').modal('show'); // 显示 modal
            $('.modal-title').text('添加设备'); // 设置 title
        }
    ),
```

增加对设备的控制功能，如能向模拟的设备发送开关命令

同样使用 Json 和 Ajax，对设备的开关属性进行修改，当设备关闭，后台不再更新设备状态并取消报警（在设备之前有报警的情况下）。

在 default 中创建 api 包，在 api 包中创建 resources.py 文件，在该文件中定义模型对应的资源类型。编辑按钮和删除按钮都是动态生成的，因此需要使用 jQuery 的动态绑定。

用户在添加和编辑设备时，若不输入设备名和设备序列号，则新建或添加失败，若不设置温度阈值，系统自动设置为 60℃。

③在这一页进行实时动态刷新，能够通过表格反映出设备的状态变化。

在本页调用 *setInterval()*方法，设置周期调用显示表格信息的函数 showListFunc()。

设备数据实时更新模块：

为实现网页的实时更新，后台需不断更新数据库。我们使用一个 python 脚本不断给服务器 POST 数据，在数据库的视图层，有相应的事件处理函数，实现对数据库的更新。

为模拟实际数据的变化，我们在对每次的数据更新时，会随机一个-3 到 3 之间的值，用当前的温度加上这一随机值，即可得到更新后的数据。

```
def add1(request):
    if request.method=='POST':
        alllist = Device.objects.filter(user=Biguser)
        for item in alllist:
            item.Temperature = str(random.randint(0, 100))
            if int(item.Temperature)>=int(item.Temrange):
                item.Alarm=True
            else:
                item.Alarm=False
            item.save()
            sn = item.SN
            tem = item.Temperature
            ut =  item.Updatetime
            ran = item.Temrange
            if(int(tem)>int(ran)):
                alr="ON"
            else:
                alr="OFF"
            todo = history(user=Biguser, SN=sn,Temperature=tem,Updatetime=ut,Temrange=ran,Alarm=alr)
            todo.save()
        return HttpResponse(json.dumps({'name':'hxq','check':1}))
```

④在这一页添加历史纪录查看接口，进入每一个设备的历史记录页。

历史记录的按钮关联进入历史记录页的接口，将该行设备所属的 id 通过 post 方式发给数据库请求调用该 id 的历史纪录，之后转跳历史纪录页，并加载数据库发来的相应设备的历史记录信息。

4、历史记录页

在这一页显示出相应设备的历史纪录。

详细设计见上一节。加载数据库发来的相应设备的历史纪录，并通过表格显示出来。

5、告警记录页

在这里生成过去时间内产生的告警记录报告，查看所有告警记录信息。

直接向数据库请求日志记录中出现报警状态的设备的记录，并加载到表格中。

二、服务器

能够处理来自前端的请求，包括页面跳转等；能够实现前端对数据库的访问和修改操作；能够处理来自模拟设备的上报状态操作。

处理简单的交互逻辑。如图，在 urls.py 中设置所有操作对应的 url。

```
v1_api = Api(api_name='v1')
v1_api.register(DeviceResource())

urlpatterns = [
    url(r'^add1/$', views.add1),
    url(r'^$', views.my_login), #登陆
    url(r'^logout/$', views.my_logout),#注销
    url(r'^index$', views.index),#主页
    url(r'^tablePage$', views.table),#显示主页设备表格
    url(r'^getlog$',views.getlog),#显示日志信息


    url(r'^api/table/$', views.showList),       #显示设备表格
    url(r'^alertlog$',views.alertlog),          #显示告警记录
    url(r'^api/del/$', views.dellist),          #删除设备
    url(r'^api/add/$', views.addlist),          #添加设备
    url(r'^api/edit/$', views.editlist),        #编辑设备
    url(r'^api/switch/$', views.switchdevice),  #设备开关操作
    url(r'^api/getlog$',views.getlog),          #获得日志页日志信息
    url(r'^api/indexlog$',views.indexlog),      #获得主页日志表格
    url(r'^api/showDevNum$',views.showDevNum),        #显示所有设备数量
    url(r'^api/showAlrDevNum$',views.showAlrDevNum),  #显示异常设备数量
    url(r'^api/showLogNum$',views.showLogNum),        #显示日志数量
    url(r'^api/showNorDevNum$',views.showNorDevNum),  #显示正常设备数量
    url(r'^getDevTemNum$',views.getDevTemNum),

    url(r'api/', include(v1_api.urls))
]
```

在 views 中实现服务器具体行为（对前端、数据库的操作和交互）。

一些重要的函数在下面给出并进行解释。

```
def showList(request):
    global Biguser
    infolist = Device.objects.filter(user=Biguser)
    info = []
    for item in infolist:
        temp = {"id": item.id, "Name": item.Name, "SN": item.SN, "Temperature": item.Temperature, "Temrange": item.Temrange, "Updatetime": item.Updatetime,
"Alarm": item.Alarm, "Operation": item.Operation}
        info.append(temp)
    res = {"info": info}
    print(info)
    return JsonResponse(res)
```

showList 函数从数据库中的 Device 表中查找将每位用户的设备信息，通过 json 串的形式返回给前端。

```
def addlist(request):
    name = request.POST['Name']
    sn = request.POST['SN']
    tem = '20'
    temrange = request.POST['Temrange']
    alarm = 0
    #uptime = datetime.datetime.now()
    op = 'ON'
    global Biguser
    Device.objects.create(user=Biguser,Name=name, SN=sn, Temperature=tem, Temrange=temrange,
                          Alarm=alarm,  Operation=op)
    res = {"success": "true"}
    return JsonResponse(res)
```

addlist()函数实现设备表格中添加设备的功能。前端录入完信息后 post 发送给服务器，服务器在数据库中添加相应记录。

```
def editlist(request):
    id = request.POST['id']
    name = request.POST['Name']
    sn = request.POST['SN']
    temrange = request.POST['Temrange']
    device = Device.objects.filter(user=Biguser).get(id=id)
    device.Name = name
    device.SN = sn
    device.Temrange = temrange
```

```
    device.save()
    res = {"success": "true"}
    return JsonResponse(res)

#@login_required()
def dellist(request):
    id = request.GET['id']
    Device.objects.filter(user=Biguser).get(id=id).delete()
    res = {"success": "true"}
    return JsonResponse(res)
```

编辑和删除设备。原理与 addlist()基本相同。

```
def switchdevice(request):
    id = request.GET['id']
    device = Device.objects.filter(user=Biguser).get(id=id)
    if device.Operation == 'ON':
        device.Operation = 'OFF'
    elif device.Operation == 'OFF':
        device.Operation = 'ON'
    device.save()
    res = {"success": "true"}
    return JsonResponse(res)
```

设备的开/关操作。

另外包括一些简单的服务器端处理函数，这里就不一一列举。他们用来实现将数据库的信息通过 json 串的形式发送给前端，前端通过表格、图表的形式表现出来。这些信息包括：设备信息、设备状态、日志信息、告警信息等。

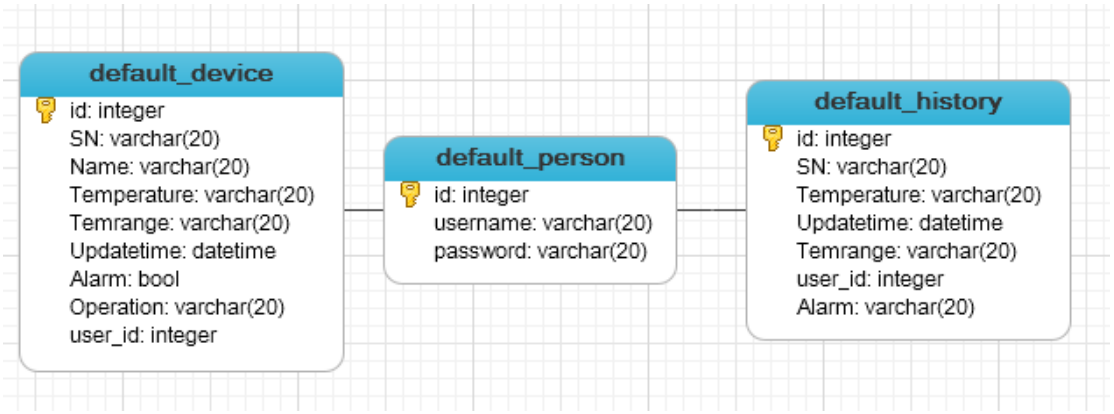
```
def getDevTemNum(request):
    temp1 = Device.objects.filter(user=Biguser)
    count1=0
    count2=0
    count3=0
    count4=0
    count5=0
    for tem in temp1:
        if (int(tem.Temperature) > 0 and int(tem.Temperature) <= 20):
            count1=count1+1
        elif (int(tem.Temperature)  >20 and int(tem.Temperature) <= 40):
            count2=count2+1
        elif (int(tem.Temperature) > 40 and int(tem.Temperature) <= 60):
            count3=count3+1
        elif (int(tem.Temperature) > 60 and int(tem.Temperature) <= 80):
            count4=count4+1
        else:
            count5=count5+1
    print('试一下')
    dict = {"catagories": ["0-20℃","20-40℃","40-60℃","60-80℃","80-100℃"],"data":[count1, count2, count3, count4, count5]}
    return JsonResponse(dict)
```

这个方法是将数据库中设备的温度信息读取出来并进行统计分类，将不同温度段的设备数量统计出来并发给前端。前端调用 ECharts 的 API 将其以柱形图的形式显示出来。

三、数据库

本系统的数据库中使用了三个表，分别为 default_device，default_person，default_history。default_person 主要用于用户管理系统使用，同时作为另两个表的外键，标记设备和历史记录，所属的用户，实现多用户管理。

实现中，我们建立了一个 person 的 model，其中包含两个属性，用户名（username）和密码（password）。




```
class Person(models.Model):
    username = models.CharField(max_length=20)
    password = models.CharField(max_length=20)
```

default_device 表用于管理设备信息，包含设备的身份信息、实时温度以及所属的用户。其中包含 8 项属性，设备序列号（SN）、设备名称（Name）、设备温度（Temperature）、设备的温度报警阈值（Temrange）、设备警报标识（Alarm）、设备开关标识（Operation）、设备信息的最后更新时间（Updatetime）以及设备的所属用户（user）。其中 user 以外键的形式实现与用户管理表（default_person）的关联。服务器通过报警标识和开关标识实现对用户状态的获取、更新或修改。设备的温度用外部程序实现实时更新，而设备更新时间会在每次更新时同时更新。

```
class Device(models.Model):
    user = models.ForeignKey(Person)
    SN = models.CharField(max_length=20)
    Name = models.CharField(max_length=20)
    Temperature = models.CharField(max_length=20)
    Temrange = models.CharField(max_length=20)
    Updatetime = models.DateTimeField(max_length=20,auto_now=True)
    Alarm = models.BooleanField(max_length=10)#OFF01 , ON
    Operation = models.CharField(max_length=20)#OFF,ON
```

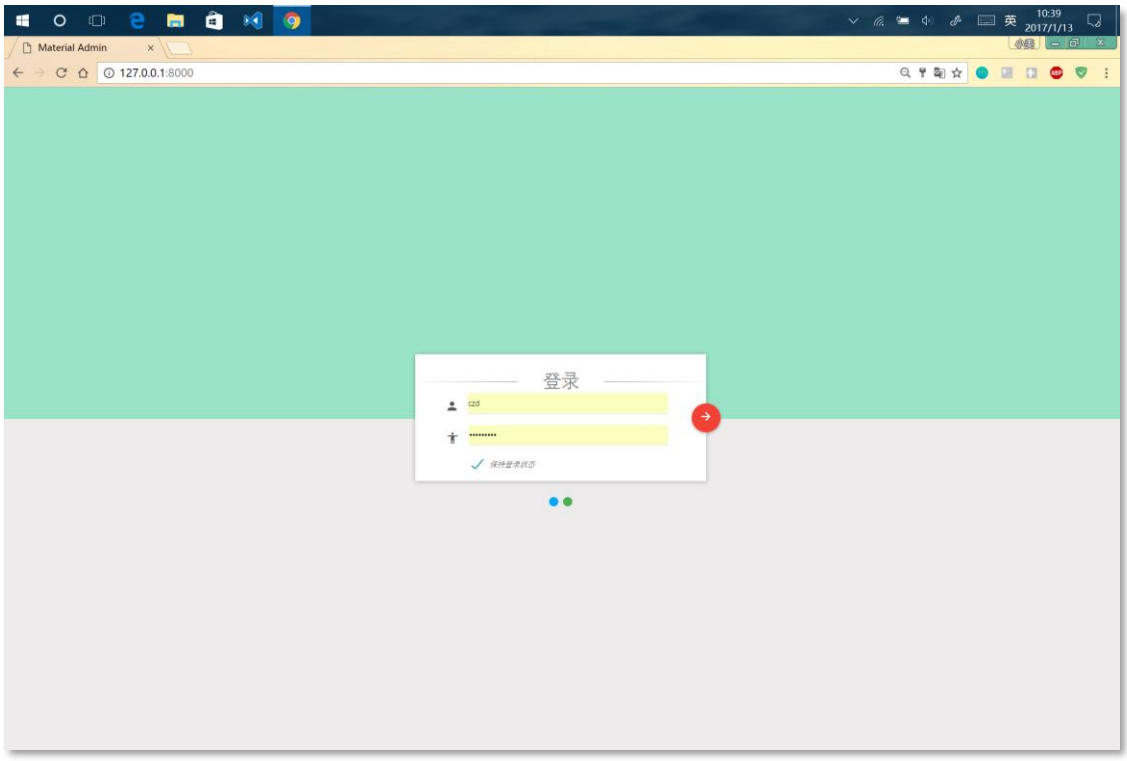
default_history 用于记录设备的各项历史记录，包括设备更新的历史记录和设备报警的历史记录。用于对设备历史记录和报警记录的显示。其中包含 6 项属性，设备序列号（SN）、设备温度（Temperature）、设备记录时间（Updatetime）、设备报警阈值（Temrange）、设备警报记录标识符（Alarm）以及历史记录所属用户信息（user）。其中 user 以外键的形式与 default_person 数据库相连，实现多用户数据信息的管理。根据设备警报记录标识符可以确定该记录为更新记录还是报警记录。设备每次更新都会将相关信息存入历史记录表。

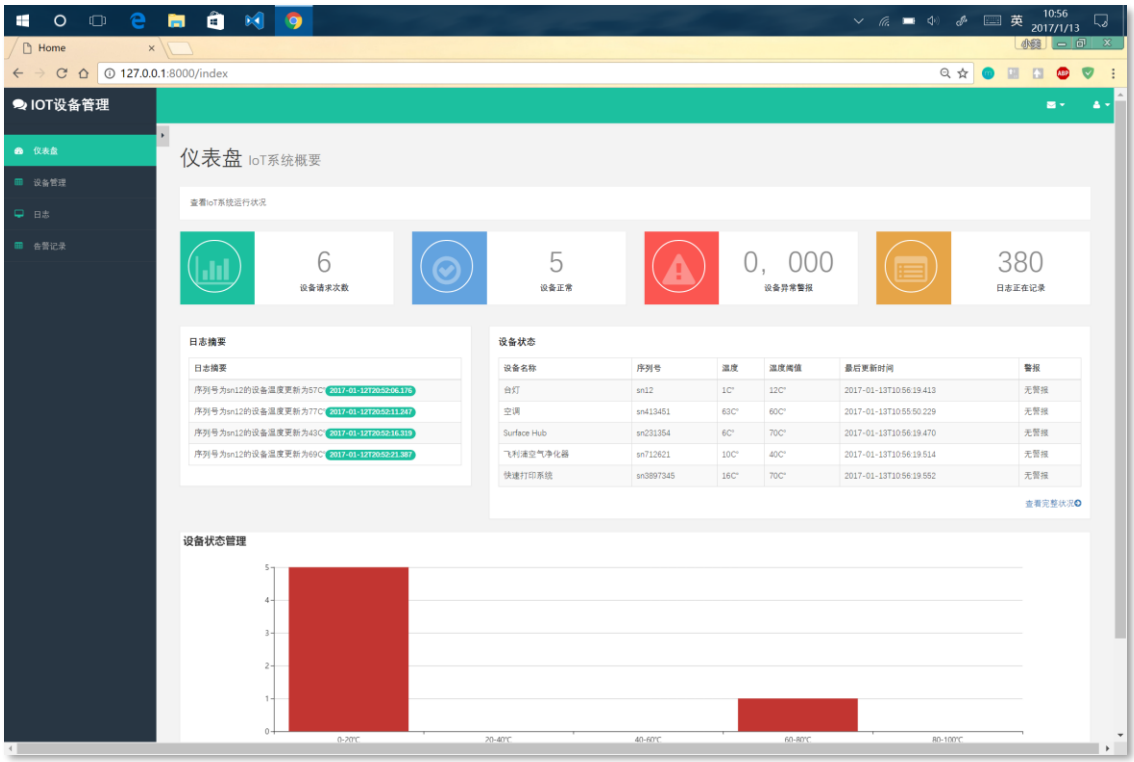
用户管理模块：

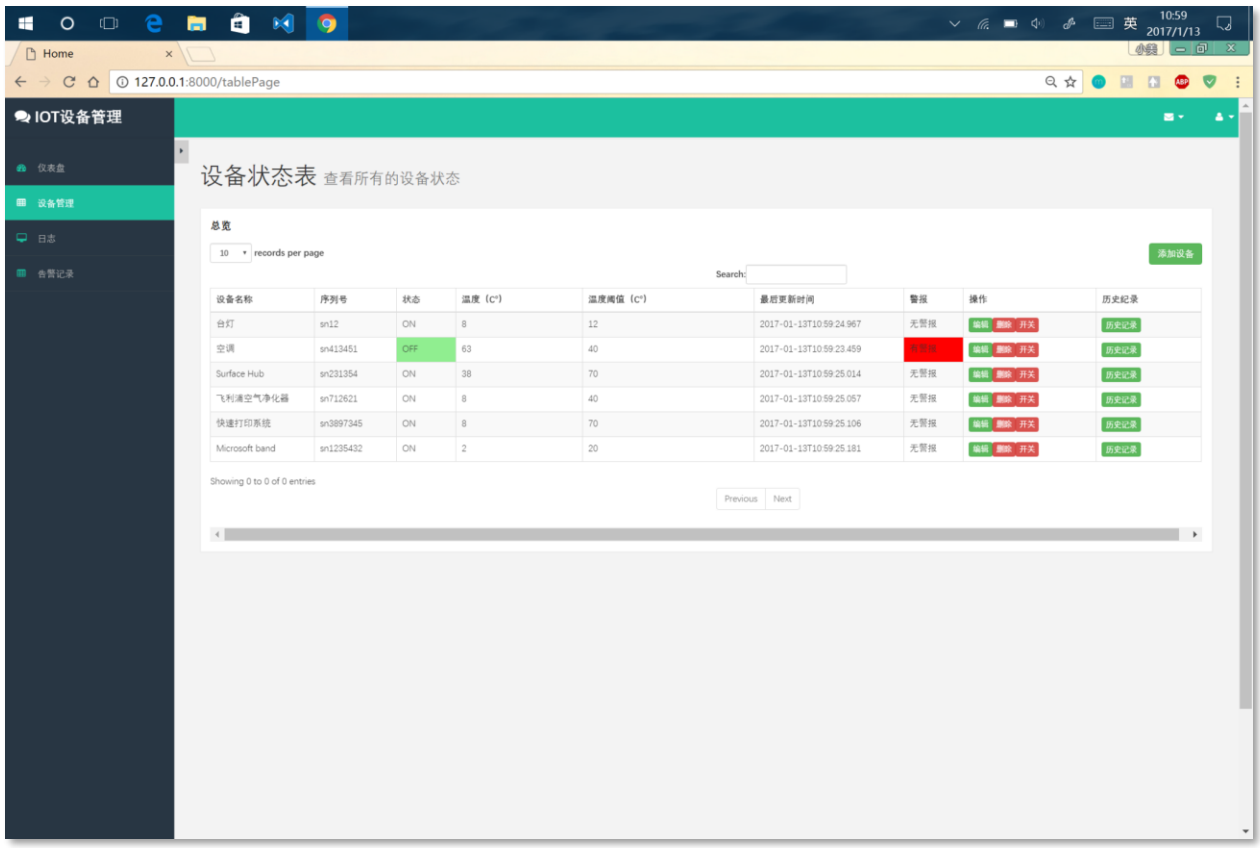
本系统的用户管理模块以自定义的 Person 类型的 model 为基础，实现用户的登陆注销以及多用户管理功能。

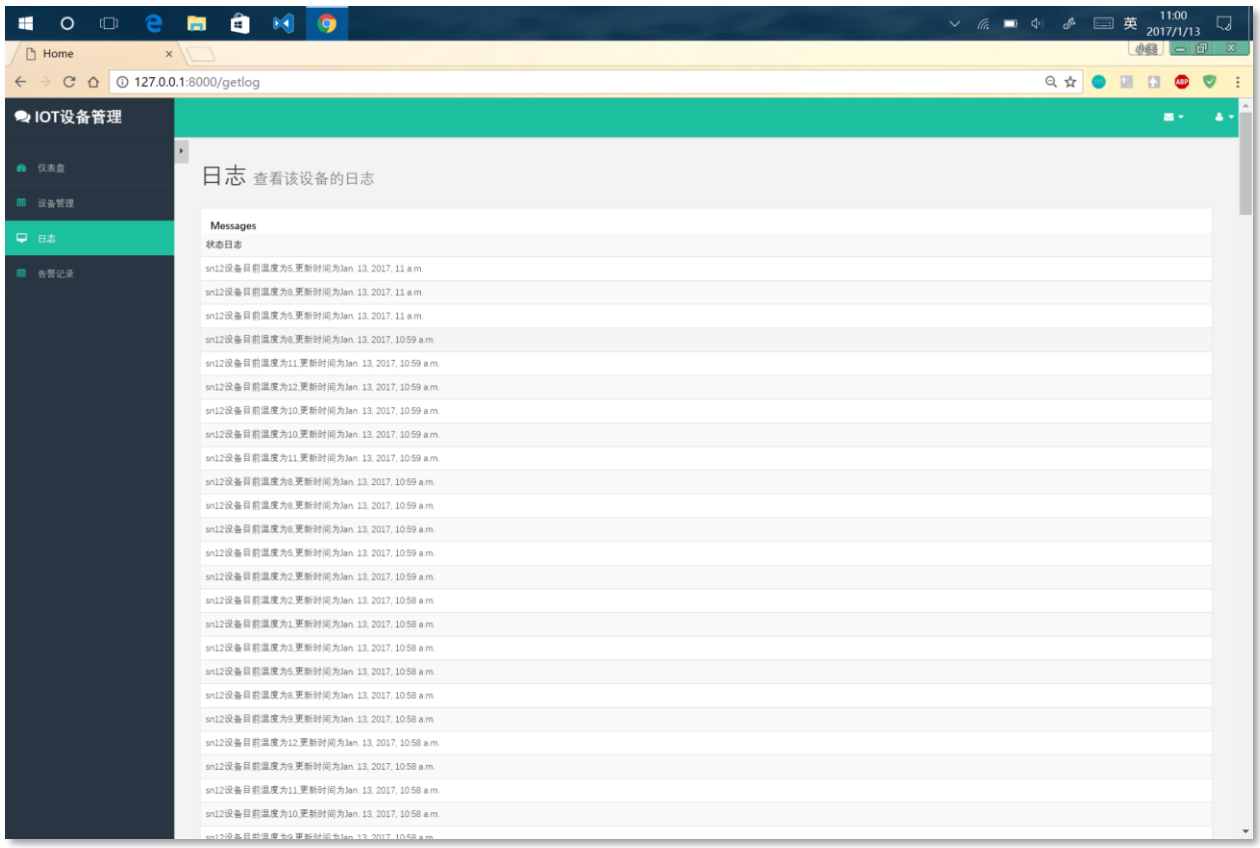
测试

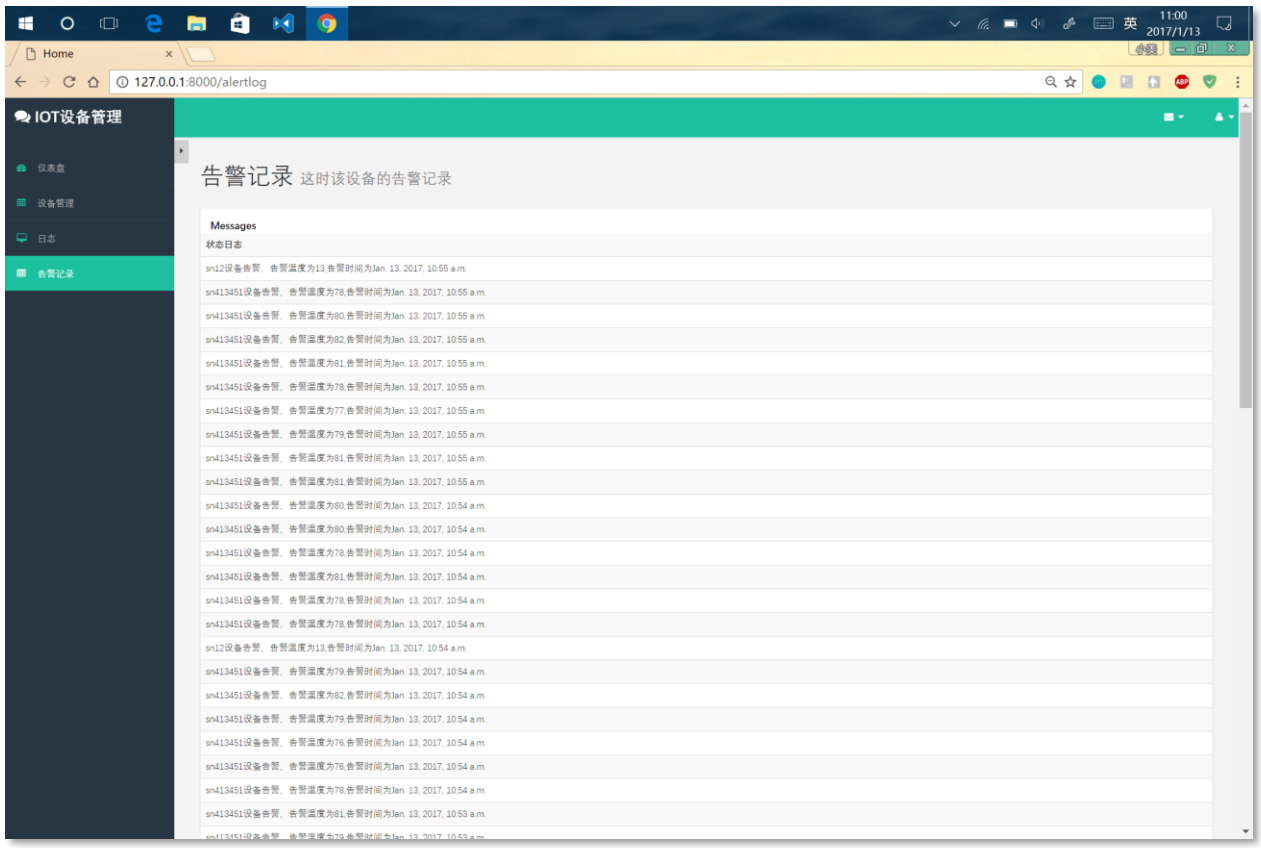
经过测试，平台能够正常工作。测试截图如下。






Home
127.0.0.1:8000/tablePage
IOT设备管理
仪表盘
设备管理
日志
告警记录
设备状态表 查看所有的设备状态
总览
10 records per page
Search:
添加设备
设备名称 序列号 状态 温度（C°） 温度阈值（C°） 最后更新时间 警报 操作 历史纪录
台灯 sn12 ON 8 12 2017-01-13T10:59:24.967 无警报 编辑 删除 开关 历史记录
空调 sn413451 OFF 63 40 2017-01-13T10:59:23.459 有警报 编辑 删除 开关 历史记录
Surface Hub sn231354 ON 38 70 2017-01-13T10:59:25.014 无警报 编辑 删除 开关 历史记录
飞利浦空气净化器 sn712621 ON 8 40 2017-01-13T10:59:25.057 无警报 编辑 删除 开关 历史记录
快速打印系统 sn3897345 ON 8 70 2017-01-13T10:59:25.106 无警报 编辑 删除 开关 历史记录
Microsoft band sn1235432 ON 2 20 2017-01-13T10:59:25.181 无警报 编辑 删除 开关 历史记录
Showing 0 to 0 of 0 entries
Previous Next


Home
127.0.0.1:8000/getlog
IOT设备管理
仪表盘
设备管理
日志
告警记录
日志 查看该设备的日志
Messages
状态日志



测试结果网站各功能能够正常使用，负载均衡。

添加设备：

点击“添加设备”按钮，弹出对话框，用户可输入设备信息，点击“Cancel”按钮取消操作，已输入信息不予记录。

点击“Save”按钮进行保存，该设备已加入设备列表中。

设备名称	序列号	状态	温度（℃）	温度阈值（℃）	最后更新时间	警报	操作	历史纪录
surface	sn10001	ON	6	50	2017-01-13T11:54:31.851	无警报	编辑 删除 开关	历史记录
Ipad	sn10003	ON	63	100	2017-01-13T11:54:32.103	无警报	编辑 删除 开关	历史记录
冰箱	sn1245	ON	1	50	2017-01-13T11:54:32.305	无警报	编辑 删除 开关	历史记录
qq	t't	ON	11	60	2017-01-13T11:54:32.521	无警报	编辑 删除 开关	历史记录
中央空调	sn2345	ON	50	50	2017-01-13T11:54:33.101	无警报	编辑 删除 开关	历史记录

点击“编辑”按钮，对该设备进行编辑。

此时可修改设备名称，设备序列号及温度阈值。

Search:

设备名称	序列号	状态	温度（℃）	温度阈值（℃）	最后更新时间	警报	操作	历史纪录
surface	sn10001	ON	1	50	2017-01-13T11:59:18.446	无警报	编辑 删除 开关	历史记录
Ipad	sn10003	ON	69	100	2017-01-13T11:59:18.816	无警报	编辑 删除 开关	历史记录
冰箱	sn1245	ON	6	50	2017-01-13T11:59:19.070	无警报	编辑 删除 开关	历史记录
qq	t't	ON	13	60	2017-01-13T11:59:19.297	无警报	编辑 删除 开关	历史记录
我的中央空调	sn1112345	ON	36	60	2017-01-13T11:59:20.199	无警报	编辑 删除 开关	历史记录

Showing 0 to 0 of 0 entries

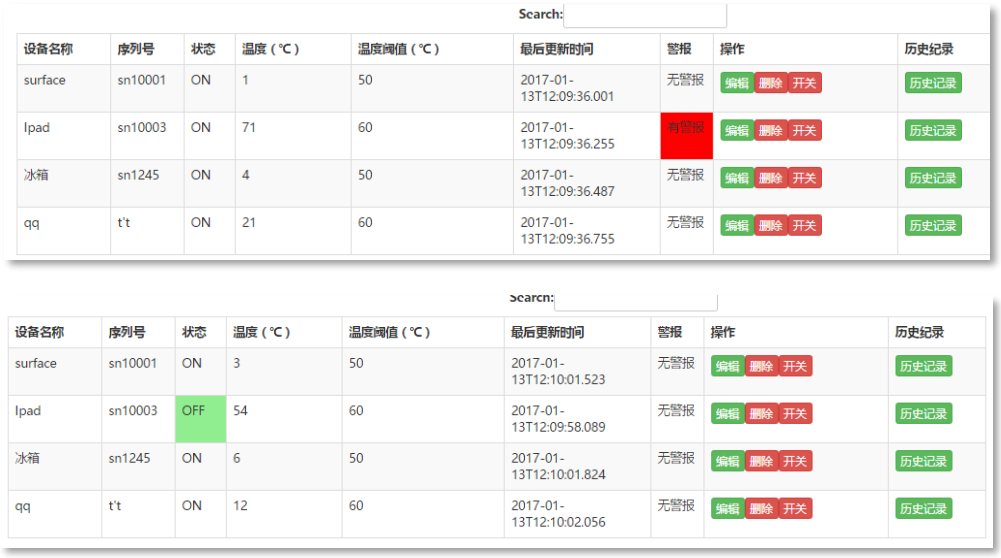
点击删除按钮，删除“我的中央空调”。

Search:

设备名称	序列号	状态	温度（℃）	温度阈值（℃）	最后更新时间	警报	操作	历史纪录
surface	sn10001	ON	0	50	2017-01-13T12:00:15.733	无警报	编辑 删除 开关	历史记录
Ipad	sn10003	ON	65	100	2017-01-13T12:00:15.980	无警报	编辑 删除 开关	历史记录
冰箱	sn1245	ON	4	50	2017-01-13T12:00:16.485	无警报	编辑 删除 开关	历史记录
qq	t't	ON	8	60	2017-01-13T12:00:16.712	无警报	编辑 删除 开关	历史记录

Showing 0 to 0 of 0 entries

点击开关按钮，即可控制对设备的开关，设备关闭后，不再更新状态，温度变化到阈值以下，取消报警。



数据库正常使用，能够正常管理各表。记录完整。能够成功更新。

存在问题及解决方案

1、未能成功使用电子邮件功能。每次发送均失败。因此我们取消了发送电子邮件的设计，设计为统一显示告警记录。

2、在设计初期一直苦于找不到模拟物联网设备向服务器发送数据的方法，最后经老师指点，我们采用一个 python 脚本，定期向服务器的指定端口发送数据。服务器的 views.py 中设置指定的函数对应处理发过来的请求。再者，在实现多用户功能时，重点在于实现各个用户的数据隔离。可以多个用户同时访问自己的资源，互不干扰。最后我们采用外键的方式，在设备和历史记录中，附加所属用户的属性，实现多用户信息的隔离。

3、设备操作部分

首先学习 AJAX 和 JSON，并学习 todolist3 工程，在队友设计的前端模板上进行实现，实现添加和删除功能比较容易，但实现编辑功能时在弹出对话框中不能显示原来的信息，经查阅资料以及请教同学，对响应 btnedit 的函数进行修改，具体实现如下：$('[name="id"]').val($(this).parents("td").siblings("td.hidden").text());（其余信息也按此方法做修改），即可成功显示。由于设备的温度是实时更新的且出于对实际情况的考虑，用户在新建设备时设备的温度由系统自动设置为 0~温度阈值之间的整数。在加入告警功能后，发现在用户不输入温度阈值时将会出现错误，原因是此时温度阈值为 None，但在后台更新状态的时候要用当前温度与阈值作比较判断是否告警，于是对温度阈值的初始化做修改，即当用户没有设置温度阈值时系统自动设置为 60℃，至此，功能实现。