

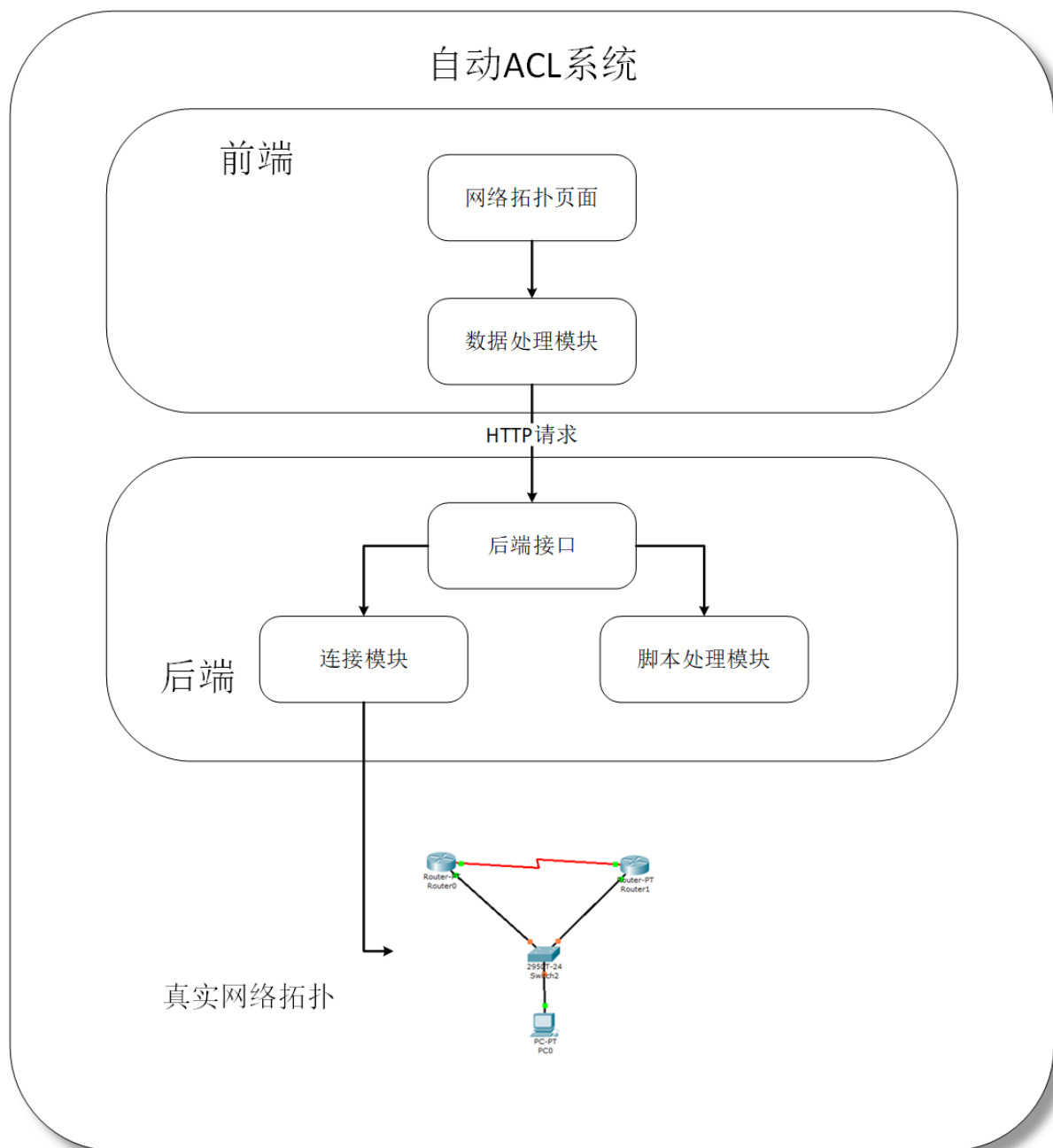
第八组第二次大作业设计文档（ACL）

1. 小组成员以及分工

姓名	学号	职责
曾康	MF20320010	组长、搭建网络环境、提供脚本、前端开发、文档编写
蔡娇娇	MF20320005	前端开发
蔡逸凡	MF20320007	前端开发
曹振飞	MF20320008	后端开发
曾彬凯	MF20320009	前端开发
陈光明	MF20320014	后端开发、文档编写

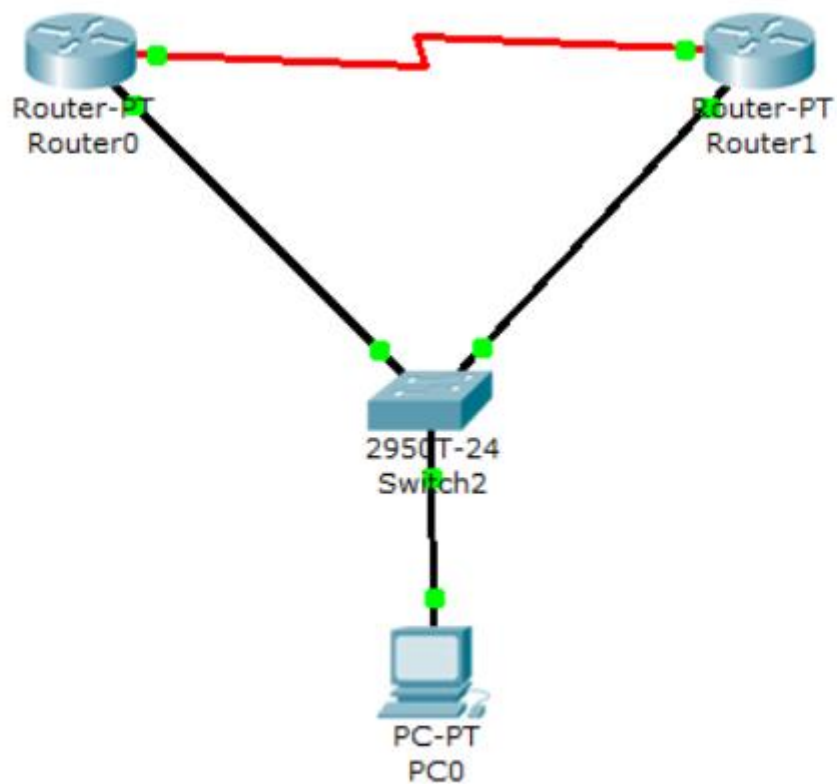
2. 软件架构设计

系统架构图如下



本系统使用 python 语言编写，前端使用静态页面构建，后端基于 Django 框架构建，前端通过 ajax 和后端进行 HTTP 通信，而后端使用 telnetlib package 与路由器进行连接及通信。

网络拓扑如下，其中包括两台路由器和一个交换机，而主程序在最下方的 PC 中运行。



3. 软件体系结构设计

3.1 前端包结构

3.2 前后端接口

接口名	method	参数	简介
/init_config	GET	routing: 路由名 hostname: 主机名 ip_address: 路由器ip	初始化路由器配置

		mask: 掩码	
/acl_config	GET	routing: 路由名	自动化配置 ACL
/cancel_acl_config	GET	routing: 路由名	清空 ACL 配置
/verify	GET	routing: 路由名	验证是否配置成功

3.3 后端文件结构

文件(夹)名	主要功能
/Telnet_Backend/script	存放发送给路由器执行的命令文本文件
/Telnet_Backend/service.py	主要功能实现文件
/Telnet_Backend/urls.py	配置接口文件
/Telnet_Backend/views.py	构建 views 的文件

4. 软件详细设计与实现

4.1 连接路由器的实现

使用 Python 中的 telnetlib 库去连接路由器。具体实现步骤：在页面初始化的时候调用连接方法，连接方法创建并保存所有需要连接的路由器的连接实例：

```
def index(request):  
    service.connect()  
    return render(request, 'index.html')
```

```
def connect():  
    global RTA  
    RTA = connect_to_routing(rta_f0)  
    global RTB  
    RTB = connect_to_routing(rtb_f0)  
    print(RTA is None)  
    print(RTB is None)
```

```
def connect_to_routing(host):  
    tn = telnetlib.Telnet(host, timeout=400000)  
  
    tn.read_until(b'Password:')  
    tn.write(password + b'\n')  
    # 进入enable模式  
    tn.write(b'enable\n')  
    tn.write(password + b'\n')  
    return tn
```

4.2 初始化配置路由器的实现

实现：前端填写某个路由器的主机名、ip 和掩码并调用后端提供的 init_config 方法，后端方法读取具体的配置命令，用前端给的参数替换命令中的占位词，最后调用 send_commands 方法将命令发送给具体的路由器去执行：

```
config.txt x
1      conf terminal
2      hostname ${hostname}
3      interface s0/0/0
4      ip address ${ip_address} ${mask}
5      no shutdown
6      end
```

```
# 初始化配置：路由器的hostname, ip地址和子网掩码
def init_config(request):
    request.encoding = 'UTF-8'
    if request.method == 'GET':
        routing = request.GET.get('routing')
        hostname = request.GET.get('hostname')
        ip_address = request.GET.get('ip_address')
        mask = request.GET.get('mask')

        print('routing' + routing)

    return HttpResponse(service.init_config(routing, hostname, ip_address, mask))

def init_config(routing, hostname, ip_address, mask):
    global rta_s0, rtb_s0

    if routing == 'RTA':
        rta_s0 = ip_address
    else:
        rtb_s0 = ip_address

    commands = read_commands("config.txt", hostname, ip_address, mask)
    return send_commands(commands, routing, 1)
```

```

# 读取文件中的命令，并替换掉其中的占位词
# 按顺序使用args中的元素替换
# 返回命令列表
def read_commands(filename, *args):
    file_path = os.path.join(os.path.abspath('.'), 'Telnet_Backend', 'script', filename)
    with open(file_path, 'r') as f:
        content = f.read()
        matches = re.findall(r"(?<=\${}).*?(?=\})", content)
        for i in range(len(matches)):
            content = content.replace("${" + matches[i] + "}", str(args[i]))
        res = content.split('\n')
    return res

```

```

# 发送命令到路由器
def send_commands(commands, routing, time_interval):
    tn = RTA if routing == 'RTA' else RTB

    tn.read_very_eager()

    for one_command in commands:
        tn.write(bytes(one_command, encoding='UTF-8') + b'\n')

    time.sleep(time_interval)
    return tn.read_very_eager()

```

4.3 自动化配置 ACL 的实现

实现：前端调用后端提供的接口方法，后端方法读取具体的配置命令，调用

send_commands 方法将命令发送给路由器去执行：

```

acl.txt x
1  conf t
2  access-l 100 deny icmp host ${ip_address1} host ${ip_address2}
3  access-l 100 permit icmp any any
4  interface s0/0/0
5  ip access-group 100 in
6  end

```

```
# 自动化配置acl
def acl_config(request):
    request.encoding = 'UTF-8'
    if request.method == 'GET':
        routing = request.GET.get('routing')

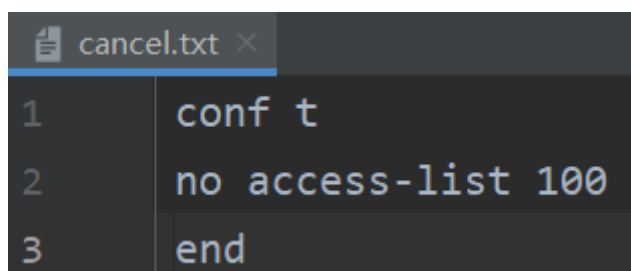
        return HttpResponse(service.acl_config(routing))
```

```
def acl_config(routing):
    rt1_s0 = rtb_s0 if routing == 'RTA' else rta_s0
    rt2_s0 = rta_s0 if routing == 'RTA' else rtb_s0

    commands = read_commands("acl.txt", rt1_s0, rt2_s0)
    return send_commands(commands, routing, 1)
```

4.4 取消 ACL 配置的实现

实现：前端点击需要取消 ACL 配置的路由器对应的按钮，调用后端 cancel_acl_config 方法，后端方法读取具体的命令并发送给具体的路由器去执行：



```
cancel.txt ×
1 conf t
2 no access-list 100
3 end
```

```
# 清空acl配置
def cancel_acl_config(request):
    request.encoding = 'UTF-8'
    if request.method == 'GET':
        routing = request.GET.get('routing')

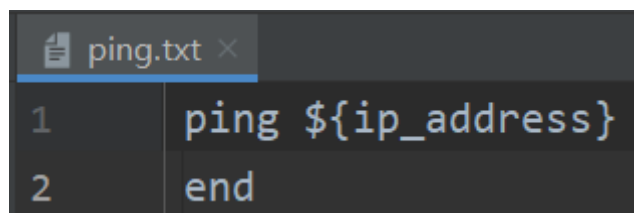
        return HttpResponse(service.cancel_acl_config(routing))
```



```
def cancel_acl_config(routing):
    commands = read_commands("cancel.txt")
    return send_commands(commands, routing, 1)
```

4.5 验证 ACL 的实现

实现：前端点击需要验证 ACL 配置的路由器对应的按钮，调用后端 verify 方法，后端方法读取具体的命令并发送给具体的路由器去执行：



A screenshot of a text editor window titled 'ping.txt'. It contains two lines of text: '1 ping \${ip_address}' and '2 end'.

```
# 验证是否配置成功
def verify(request):
    request.encoding = 'UTF-8'
    if request.method == 'GET':
        routing = request.GET.get('routing')

        return HttpResponse(service.verify(routing))
```

```
def verify(routing):
    rt_s0 = rta_s0 if routing == 'RTB' else rtb_s0
    commands = read_commands("ping.txt", rt_s0)
    result = send_commands(commands, routing, 5)
    print(result)
    return result
```