Design&Core Code

I select 15(7 from the testing program and 8 from the management platform) functions to illustrate with the introductions and the core codes.

一、The testing program section

1.the optimalization of the socket:

In order to increase efficiency,there are two threads during testing the IP. The sending thread reads the IP from the database file and builds and sends the ICMP packet. The receiving thread monitors and receives the ICMP packets and reads its IP.

Core code：

The sending thread：

```
......
cu.execute("select * from default_todolist")
    for item in cu.fetchall():
        the_id,dest_addr,n = item
        my_socket.sendto(packet, (dest_addr, 1))
       ......
```

The receiving thread：

```
......
whatReady = select.select([my_socket], [], [], timeLeft)
        if whatReady[0] == []: # Timeout
            break
        else:
            recPacket, addr = my_socket.recvfrom(1024)
            add,m=addr
           ......
```

2.communication between the management platform and the testing program

The testing program and the management platform communicate with the TCP socket. It tests the IP once received the TCP, or tests the IP every 30 seconds. The testing program is the server.

Core code：

The testing program（as the server）：

```
    address = ('localhost', 9999)
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind(address)
    s.listen(5)

    while 1:
        ...
        whatReady = select.select([s], [], [], 27)
```

```
            s.close()
            s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            s.bind(address)
            s.listen(5)
```

The management platform（as the client）:

```
def refresh(request):
      address = ('localhost', 9999)
      s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
      s.connect(address)
      s.send('refresh')
      s.close()
     .......
```

3.the optimalization of the threads：
The sending and the receiving thread are created when the TCP signals come or every 30 seconds. The sending thread terminates automatically once the packets have been sent. The receiving thread judges the remaining IP to be inaccessible, when it has not received ICMP packets for 2 seconds. The main function monitors the IP with the select function in an infinite loop.

Core code：
Main function：

```
......
while 1:
            thread.start_new_thread( receive_one_ping, (my_socket, my_ID, 2) )
            thread.start_new_thread(  send_one_ping,  (my_socket,  "www.baidu.com", 
my_ID) )
            time.sleep(3)
            whatReady = select.select([s], [], [], 27)
           ......
```

4.Build the ICMP packet:
Core code：

```
     header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, 0, my_checksum, ID, 
1)
      bytesInDouble = struct.calcsize("d")
      data = (192 - bytesInDouble) * "Q"
      data = struct.pack("d", time.time()) + data
      #calculate the checksum
      my_checksum = checksum(header + data)
      header = struct.pack(
            "bbHHh", ICMP_ECHO_REQUEST, 0, socket.htons(my_checksum), ID, 1
      )
      packet = header + data
```

5.The checksum function:

Core code:

```
def checksum(source_string):
        sum = 0
        countTo = (len(source_string)/2)*2
        count = 0
        while count<countTo:
                thisVal = ord(source_string[count + 1])*256 + ord(source_string[count])
                sum = sum + thisVal
                sum = sum & 0xffffffff
                count = count + 2

        if countTo<len(source_string):
                sum = sum + ord(source_string[len(source_string) - 1])
                sum = sum & 0xffffffff

        sum = (sum >> 16)    +    (sum & 0xffff)
        sum = sum + (sum >> 16)
        answer = ~sum
        answer = answer & 0xffff

        answer = answer >> 8 | (answer << 8 & 0xff00)

        return answer
```

6.the history data function

The receiving thread receives a packet and reads its IP. Then the thread will do this:

```
if(the IP is in the default_todolist table)
       if(the IP is in the default_historydata table)
              if(the state of the IP has changed)
                     create a new record
else
       create a new record
```

Core code:

```
                    ......
                    add,m=addr
                       cu.execute("select * from default_todolist where ip=(?)",(add,))
                       the_arrive_ip=cu.fetchall()
                       if the_arrive_ip:
                              iplist.remove((add,))
                              cu.execute("update      default_todolist      set      state=1      where
          ip=(?)",(add,))
                              cx.commit()
                              cu.execute("select * from default_historydata where ip=(?) order by time desc",(add,))
                              items=cu.fetchall()
                              if items:
                                   ladb2_item=items[0]
                                   item_id,item_add,item_time,item_state=ladb2_item
                                   if item_state==0:
                                        cu.execute("select * from default_historydata")
                                        items2=cu.fetchall()
                                        t = (len(items2)+1,add,time.time(),1)
                                        cx.execute("insert into default_historydata values (?,?,?,?)", t)
                                        cx.commit()
                              else:
                                   cu.execute("select * from default_historydata")
                                   items2=cu.fetchall()
                                   t = (len(items2)+1,add,time.time(),1)
                                   cx.execute("insert into default_historydata values (?,?,?,?)", t)
                                   cx.commit()
                                ......
```

7.mechanisms to judge whether it is inaccessible：
When the receiving thread is created, it will create a list of all the IP in the database file. Then it will remove the IP it receives. When the timeout occurs, the remaining IP will be considered as inaccessible.
Core code：

```
             ......
             add,m=addr
               cu.execute("select * from default_todolist where ip=(?)",(add,))
               the_arrive_ip=cu.fetchall()
               if the_arrive_ip:
                    iplist.remove((add,))
                    ......
     for i in iplist:
          cu.execute("update default_todolist set state=0 where ip=(?)",i)
          cx.commit()
        ......
```

8.The data-sharing between the management platform between the testing program：
The management platform creates the database. The testing program runs and

modifies the database file.

二、The management platform section

1.Just delete the IP with partially refresh with AJAX

Core code:

```
    $(document).ready(
            $("#tabProduct").on('click', ".btnDel", function () {
                        $.ajax({
                            url: "/del/",
                            type: "GET",
                            data:                                    {"todoid":
$(this).parent("td").siblings("td.hidden").text()},
                            dataType: "JSON",
                            success: function (data) {
                                reload_table();
                            },
                            error: function (jqXHR, textStatus, errorThrown) {
                                reload_table();
                                alert('error during deleting！ ');
                            }
                        })
                }
            ),
            reload_table()
    )
```

JavaScript section（内嵌在 getlist.html 中）:

```
    function reload_table() {
        $.ajax({
            url: "/todogetlist/",
            type: "get",
            dataType: "JSON",
            success: function (data) {
                $("#tabProduct").children("tbody").empty()
                var htmlstr = ""
                for (var i = 0; i < data.todolist.length; i++) {
                    htmlstr = htmlstr + "<tr>\n" +
                            "<input          type=\"hidden\"          value=\"\"
name=\"id\"/>\n" +
                            ......
                            "<a href=\"/history/?todoid=" + data.todolist[i].ip +
"\" type=\"button2\" name=\"Submit2\">see the history data</a>\n" +
                            "</td>\n" +
                            "</tr>";
                }
```

```
                    $("#tabProduct").children("tbody").html(htmlstr);
                },
                error: function (jqXHR, textStatus, errorThrown) {
                    alert('error during gaining the data');
                }
            });
            setTimeout(reload_table, 5000);
        }
```

Views.py section：

```
@login_required
def dellist(request):
    id = request.GET['todoid']
    Todolist.objects.filter(ip=id).delete()
    res ={"success":"true"}
    return JsonResponse(res)

@login_required
def todogetlist(request):
    ......
    res ={"todolist":todolist}
    return JsonResponse(res)
```

2.Refresh the web partially with AJAX every 5 seconds

```
    function reload_table() {
        $.ajax({
            ......
        });
        setTimeout(reload_table, 5000);
    }
```

3.Add the network segment

Input an IP and the length of the network segment to add all the IP in this network segment

Core code：

```
f3= request.POST['field3']
packedIP = socket.inet_aton(f3)
ip_num=struct.unpack("!L", packedIP)[0]
f4=request.POST['field4']
netlength = 32-int(f4)
power=pow(2,netlength)
ip_num=ip_num/power
ip_num=ip_num*power
```

```
j=0
while (j<power):
        ip_ip=socket.inet_ntoa(struct.pack('!L',ip_num))
        todo = Todolist(ip=ip_ip,state=0)
        todo.save()
        j=j+1
       ip_num=ip_num+1
```

4.login function:
Core code：
```
views.py:
.......
@login_required
def getlist(request):
        ...

@login_required
       def addlist(request):
       ...
......
```

5.classification user purview：
Core code：
```
Views.py:
if user.is_superuser:
        return redirect('/')
else:
        return redirect('/getlist2')

Url.py:
url(r'^getlist2/todogetlist/$',"default.views.todogetlist"),
url(r'^getlist2/$', 'default.views.getlist2'),
```

6.the logout function in views.py
Core code：
```
@login_required
def logout_view(request):
        logout(request)
        return redirect('/account/login')
```

7.Format validation for IP with JavaScript regex
Core code：
```
var Regx = /^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$/;
```

8.The alert for error password

Core code：

```
alert("error password");
```