1. iptables的四表五链中，表的概念是什么，有哪些表，作用是什么，链的概念是什么，有哪些链，作用是什么

```
# 链
Netfilter是内核的数据包处理框架，他通过5个hook点在数据包的不同处理阶段对其进行操作
这5个hook点分别是

PREROUTING      --- 到达
INPUT           --- 输入
FORWARD         --- 转发
OUTPUT          --- 输出
POSTROUTING     --- 发送

分别在数据包到达，输入，转发，输出和发送的阶段触发，
用户可以通过iptables在用户空间配置规则，这些规则定义了如何在各个钩子点处理这些数据包。这些规则通过iptables工具传递给Netfilter，并由Netfilter在适当的钩子点上应用

# 表
表（tables） 是逻辑上的分组，用于组织不同类型的规则

filter: 负责对数据包进行过滤
nat:  主要负责数据包的网路地址转换，主要包括：SNAT，DNAT
mangle：主要用于修改数据包的内容
security: hatred-selinux 略
```

2. iptables中表与链之间的关系是怎样的

```
每个钩子点可以引用多个表中的规则链，从而实现对数据包的多方面处理。例如：

PREROUTING 钩子
      可以引用 nat 表中的 PREROUTING 链来进行 DNAT 操作，
      也可以引用 mangle 表中的 PREROUTING 链来修改数据包属性。
INPUT 钩子
      可以引用 filter 表中的 INPUT 链来实现防火墙过滤，
      也可以引用 mangle 表中的 INPUT 链来修改包的某些属性。
OUTPUT 钩子
      可以引用 nat 表中的 OUTPUT 链来进行地址转换，
      也可以引用 filter 表中的 OUTPUT 链来实现流量控制。
```

3. 通过iptables抓取TCP三次握手的SYN_SENT状态和SYN-REV状态

```
#10.0.0.133 作为服务端
apt install -y nginx

cd /var/www/html
dd if=/dev/zero of=test.img bs=1M count=100
```

```
iptables -t filter -A INPUT -s 10.0.0.132 -d 10.0.0.133 -p tcp --tcp-flags
SYN,ACK,FIN,RST SYN -j DROP

# 10.0.0.132 作为客户端
[root@ubuntu2204 ~]#curl --limit-rate 1k -O  http://10.0.0.133/test.img
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:--  0:02:10 --:--:--     0

# 可观察到SYN-SENT
[root@ubuntu2204 ~]#ss -nta
SYN-SENT                       0                             1
                                 10.0.0.132:53438
              10.0.0.133:80


#################################################################################
##################################

# 10.0.0.132
iptables -t filter -A INPUT -s 10.0.0.133 -d 10.0.0.132 -p tcp --tcp-flags
SYN,ACK,FIN,RST SYN,ACK -j DROP

[root@ubuntu2204 ~]#curl --limit-rate 1k -O  http://10.0.0.133/test.img
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:--  0:02:10 --:--:--     0

# 10.0.0.133
[root@ubuntu2204 html]#ss -nta
State                          Recv-Q                         Send-Q
                               Local Address:Port
            Peer Address:Port                         Process
LISTEN                         0                              128
                                 127.0.0.1:6010
               0.0.0.0:*
LISTEN                         0                              511
                                     0.0.0.0:80
               0.0.0.0:*
LISTEN                         0                              4096
                               127.0.0.53%lo:53
                 0.0.0.0:*
LISTEN                         0                              128
                                     0.0.0.0:22
               0.0.0.0:*
SYN-RECV                       0                              0
                                  10.0.0.133:80
            10.0.0.132:49478
ESTAB                          0                              52
                                 10.0.0.133:22
                10.0.0.1:41134
```

4. SNAT是在什么链上工作，为什么？DNAT是在什么链上工作，为什么？

```
# DNAT（Destination NAT）用于修改数据包的目标地址（destination address）。

而路由决策依赖目标地址
在路由阶段，系统需要知道数据包的目标地址来决定将其转发到哪个接口。
因此，DNAT 必须在路由决策之前完成，这就是为什么 DNAT 在 PREROUTING 链 上发生。

# 源地址修改： SNAT（Source NAT）用于修改数据包的源地址（source address）
路由决策与源地址无关：
路由决策只依赖目标地址，因此 SNAT 可以在路由完成之后进行。
在 POSTROUTING 阶段，数据包已经决定了从哪个接口发出，修改源地址不会影响路由决策。
路由决策前修改源地址，可能导致不必要的路由冲突或复杂性。

路由表依赖真实的源地址信息
如果在路由决策前修改了源地址，路由表的行为可能与预期不一致。例如：
假设路由表基于某些策略（如源地址路由）决定数据包的路径。
修改源地址后，路由表可能选择错误的路径或接口
```

5. **问题**：假设有一个服务需要在不同操作系统上安装不同的软件包：
   - Ubuntu 上需要安装 `nginx`。
   - CentOS 上需要安装 `httpd`。
   - 如何编写一个任务，动态选择需要安装的软件包？

   任务：编写一个 Playbook，利用变量、`when` 条件和 `ansible_facts` 动态选择软件包并安装。

```yaml
- hosts: all
  gather_facts: no
  remote_user: root
  tasks:
  - name: filter var
    setup:
      #filter:
      gather_subset: min
    register: info

#  - name: print info
#    debug:
#      var: info["ansible_facts"]["ansible_distribution"]

  - name: install webserver
    apt:
      name: nginx
      state: present
    #debug:
    #  msg: "install nginx"
    when: info["ansible_facts"]["ansible_distribution"] == "Ubuntu"

  - name: install httpd
```

```
    yum:
      name: httpd
      state: present
    #debug:
    #  msg: "install httpd"
     when: info["ansible_facts"]["ansible_distribution"] == "Rocky"
```

6. 使用ansible，写个role做k8s环境初始化（仅满足Ubuntu即可，无需考虑多版本问题），k8s环境初始化包括

```
# K8s集群分为3种角色
master：10.0.0.11， 10.0.0.12， 10.0.0.13
node：10.0.0.14， 10.0.0.15， 10.0.0.16
haproxy: 10.0.0.17, 10.0.0.18

将上述的8个机器
初始化网卡 ---> 将DNS指向私有DNS10.0.0.3
关闭swap
关闭防火墙
设置时间同步
更改hostname
  - master的hostname改为：master+ip后缀+[URL] --> master11.feng.org, master12.feng.org, master13.feng.org
  - node的hostname改为：node+ip后缀+[URL] --> node14.feng.org, node15.feng.org, node16.feng.org
  - haproxy的hostname改为：master+ip后缀+[URL] --> haproxy17.feng.org, haproxy18.feng.org


# [选做]
并将hostname和IP的对应关系写入/etc/hosts,hosts文件最终实现如下
10.0.0.11 master11.feng.org
10.0.0.12 master12.feng.org
10.0.0.13 master13.feng.org
10.0.0.14 node14.feng.org
10.0.0.15 node15.feng.org
10.0.0.16 node16.feng.org
10.0.0.17 haproxy17.feng.org
10.0.0.18 haproxy18.feng.org
```

最终执行ansible-playbook env_init.yaml，一键完成全部配置

```
整个目录结构

├── ansible.cfg
├── hosts
├── roles
|   ├── env_init
|   |   └── tasks
|   |       ├── main.yaml
|   |       ├── network_init.yaml
```

```
|   |       ├── set_hosts.yaml
|   |       ├── stop_firewall.yaml
|   |       ├── stop_swap.yaml
|   |       └── sync_time.yaml
|   ├── haproxy_init
|   |   └── tasks
|   |       ├── haproxy_set_hostname.yaml
|   |       └── main.yaml
|   ├── master_init
|   |   └── tasks
|   |       ├── main.yaml
|   |       └── master_set_hostname.yaml
|   └── node_init
|       └── tasks
|           ├── main.yaml
|           └── node_set_hostname.yaml
└── init.yaml
```

```
# vim init.yaml
- hosts: master
  gather_facts: no
  remote_user: root
  vars:
    NETFILE_NAME: 01-netcfg.yaml
    DNS_IP: 10.0.0.3
    URLNAME: feng.org

  roles:
    - master_init

- hosts: node
  gather_facts: no
  remote_user: root
  vars:
    NETFILE_NAME: 01-netcfg.yaml
    DNS_IP: 10.0.0.3
    URLNAME: feng.org

  roles:
    - node_init

- hosts: haproxy
  gather_facts: no
  remote_user: root
  vars:
    NETFILE_NAME: 01-netcfg.yaml
    DNS_IP: 10.0.0.3
    URLNAME: feng.org

  roles:
    - haproxy_init

- hosts: all
  gather_facts: yes
```

```
  remote_user: root
  vars:
    NETFILE_NAME: 01-netcfg.yaml
    DNS_IP: 10.0.0.3
    URLNAME: feng.org

  roles:
   - env_init

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

# cat /roles/master_init/tasks/master_set_hostname.yaml
- name: Register hostname
  shell: echo master`hostname -I |awk '{print $1}' | awk -F'.' '{print $4}'`
  register: HOSTNAME_OUTPUT


- name: Set the system hostname
  shell: hostnamectl set-hostname {{ HOSTNAME_OUTPUT.stdout }}.{{ URLNAME }}

# cat /roles/master_init/tasks/main.yaml
- include_tasks: master_set_hostname.yaml

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

# cat /roles/node_init/tasks/node_set_hostname.yaml
- name: Register hostname
  shell: echo node`hostname -I |awk '{print $1}' | awk -F'.' '{print $4}'`
  register: HOSTNAME_OUTPUT

- name: Set the system hostname
  shell: hostnamectl set-hostname {{ HOSTNAME_OUTPUT.stdout }}.{{ URLNAME }}

# cat main.yaml
- include_tasks: node_set_hostname.yaml

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

# cat /roles/haproxy_init/tasks/haproxy_set_hostname.yaml
- name: Register hostname
  shell: echo haproxy`hostname -I |awk '{print $1}' | awk -F'.' '{print $4}'`
  register: HOSTNAME_OUTPUT

- name: Set the system hostname
  shell: hostnamectl set-hostname {{ HOSTNAME_OUTPUT.stdout }}.{{ URLNAME }}

# cat main.yaml
- include_tasks: haproxy_set_hostname.yaml

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

# cat /roles/env_init/tasks/network_init.yaml
```

```
- name: del search
  lineinfile:
    path: /etc/netplan/{{ NETFILE_NAME }}
    regexp: 'search'
    backup: yes
    state: absent

- name: reload network
  shell: netplan apply

- name: add DNS
  replace:
    path: /etc/netplan/{{ NETFILE_NAME }}
    regexp: '(        addresses:\s*)\[.*'
    replace: '\1[{{ DNS_IP }}]'

# cat /roles/env_init/tasks/set_hosts.yaml
- name: Collect IP and hostname
  set_fact:
    host_info: "{{ ansible_default_ipv4.address }} {{ ansible_hostname }}.
{{URLNAME}}"

- name: Gather all hosts info
  run_once: true
  set_fact:
    #all_hosts_info: "{{ groups['all'] | map('extract', hostvars, 'host_info') |
list }}"
    # group['all']ansible中的内置变量，输出所有主机，例如：hosts:["10.0.0.11",
"10.0.0.12", ..., "10.0.0.18"]
    # map('extract', hostvars, 'host_info')
    # `extract`函数用于从字典中提取指定键的值。与`hostvars`结合使用时，可以从每个主机的变量
中提取特定的量
    # select('defined') 过滤掉 host_info 未定义的主机。确保结果中只包含定义了 host_info
的主机，避免运行时错误
    # |list : 将过滤后的结果转换为标准的 Python 列表。
    all_hosts_info: "{{ groups['all'] | map('extract', hostvars, 'host_info') |
select('defined') | list }}"

- name: Debug all hosts info
  run_once: true
  debug:
    msg: "{{ all_hosts_info }}"

- name: Add all hosts info to /etc/hosts
  lineinfile:
    path: /etc/hosts
    line: "{{ item }}"
    create: yes
  with_items: "{{ all_hosts_info }}"
  when: item != ''


#cat /roles/env_init/tasks/sync_time.yaml
- name: sync time
  shell: timedatectl set-timezone Asia/Shanghai
```

```
- name: install chrony
  apt: update_cache=yes name=chrony state=present

- name: set chrony_config_file
  lineinfile:
    path: /etc/chrony/chrony.conf
    insertafter: "pool ntp.ubuntu.com        iburst maxsources 4"
    line: "pool ntp.aliyun.com        iburst maxsources 2"

- name: reload chrony
  shell: systemctl enable chrony && systemctl restart chrony

# cat /roles/env_init/tasks/stop_firewall.yaml
- name: stop firewall
  shell: systemctl disable --now ufw

# cat /roles/env_init/tasks/stop_swap.yaml
- name: stop swap
  shell: systemctl disable --now swap.img.swap && systemctl mask swap.img.swap

# cat main.yaml
- include_tasks: network_init.yaml
- include_tasks: set_hosts.yaml
- include_tasks: sync_time.yaml
- include_tasks: stop_firewall.yaml
- include_tasks: stop_swap.yaml
```