- **nginx是个什么软件？nginx四层是怎么实现的？四层代理软件还有哪些？**
- nginx可以用作web服务器，四七层代理，负载均衡，还有缓存服务器。
- nginx使用了一个stream模块来实现四层tcp。我们在stream里面配置一组server，用proxy_pass指定server名字，然后在下面再配置该server的upstream轮询，就是server加上代理服务的ip和端口，还有轮询的方式。这样就配置好了nginx四层。
- 还有lvs、haproxy等等
- **keepalived的原理是什么？**
- keepalived实现高可用是通过VRRP虚拟路由冗余协议来实现的，n台相同功能的服务器组成一个路由器组，它会产生一个虚拟ip在路由器组中按照一定规则在路由器组进行切换，vip所在的机器被称为master，其他的是backup，当前的master会定时地以多播的形式向backup发送心跳信息，如果master发生故障，backup无法接收到信息，就会根据VRRP的优先级来选举一个新的master，虚拟ip就会切换到该新的master上，从而实现高可用。keepalived有两种模式，一个是抢占模式，另一个是非抢占模式。抢占模式就是当之前的master恢复的时候，就会将VIP抢占过来，非抢占就是不会抢占，而是成为一个新的backup。我之前做keepalived时还遇到过脑裂的问题，当时我在master和backup上都发现了vip，然后在backup上使用tcpdump抓包tcpdump -i eth0 | grep VRRP也看到了VRRP数据，就觉得很奇怪，很正常呀。然后找了半天原因最后才发现是防火墙启用了，而且没有开启接收VRRP协议，导致backup收不到VRRP包，就自己上位了。然后我就想，要如何防止这种情况发生呢？就问了一下一些前辈，上网查找了资料，有说第三方仲裁的看了一下比较麻烦，后面知道了一种简单的方法，借助keepalived配置里面自带的vrrp_script以及tract_script指定运行一个脚本，去ping网关，如果不通的话，就关闭keepalived。
- **keepalived和heartbeat有什么相同点和不同点？**
- 相同点就是都是做高可用的热门工具。虽然它们目标是一样的，但是不同点有很多。keepalived的配置相对来说更简单，而heartbeat的功能更为强大，它们的协议不同，keepalived使用的是VRRP协议通过发送VRRP包来保证高可用的，而heartbeat使用的是心跳进行通信和选举，心跳还可以走串口，可以有效的防止网络带来的脑裂问题，它们都可以根据业务来自定义脚本，但是heartbeat脚本需要支持service start这种方式，而keepalived没有限制。我觉得一般来说，keepalived就可以满足大部分需求，而且它配置比较简单，在发生故障的时候解决问题的速度更快。
- **你们使用的是什么监控软件？zabbix的原理，监控方式，监控的具体步骤**
- 我们使用的监控软件是zabbix。zabbix需要一个lnmp环境，即linux+nginx+mysql+php。zabbix的agentd被安装在被监控端主机上，agent负责收集客户端本地监控数据，然后发送到server端，server端收到信息后将之存储在本

地数据库中，nginx根据数据在前端进行展现和绘图。zabbix有agent、snmp、ipmi三种监控方式。agent方式下zabbix基于自身的zabbix_agent客户端插件监控OS状态，比如cpu、内存、硬盘、网卡等。snmp方式下zabbix通过简单网络管理协议监控设备，通过设定snmp的参数将相关的数据传送到服务端。ipmi是指智能平台管理接口，主要应用于设备的物理特性，监控目标机器的温度、电压、电扇工作状态，电源供应等等。

- 我监控过nginx服务，还通过脚本监控过服务状态。首先需要在nginx配置文件里加入ngx_status开启访问功能。然后在客户端编写了一个脚本，主要用来检测nginx进程是否存在，以及检测nginx性能，写好之后chmod将脚本加上执行权限，然后修改客户端的配置文件，将unsafeuserparameters开启，在用户参数后面加上自定义的nginx.status[*]，跟上脚本的路径，这样就配置好了客户端。接下来就是服务端的监控配置了，首先创建一个主机，写上nginx的ip和agent端口。然后创建一个模板，可以选择系统自带的，也可以创建一个空的模板。模板创建完成后点击下面的监控项，选择创建监控项，将之前配置好的键值对添加进去，我当时是创建了7个监控项，也就是脚本里面对应的7个功能，包括了读写状态，等待时间，访问请求状态等等。之后就是创建图像，就将所有的勾上，在一个图像上显示就好。然后就创建触发器，比如我的等待时间就设置为了6秒黄色警告，10秒红色严重问题。表达式可以点击旁边的增加提供模板。有了触发器，肯定就要创建邮件报警。这个需要在服务端安装配置一个sendmail或者postfix服务，我当时做的是sendmail。在管理下面的示警媒介写入自己的邮件信息，然后再创建动作，将相关信息写入，启用邮件就可以通过邮件收到报警信息了。
- **网站的动静态分离是怎么做的?**
- 我们是有单独的静态域名img，单独的nginx服务器，然后通过七层nginx对域名进行判断，将静态文件的请求转发到静态服务器。具体是这样的，修改nginx的配置文件，在http模块下加入两个upstream池，分别是动态服务器池和静态服务器池，里面分别指定了动静态服务器的ip端口以及轮询算法。然后在下面指定两个server，一个server加入proxy_pass匹配动态池，另一个server加入proxy_pass以及proxy_set_head将所有访问静态域名的请求转到静态池中。由动态池和静态池利用轮询算法将请求转发到相对应的服务器上去，这样就完成了nginx的动静态分离。
- 当然，如果没有单独的静态域名的话，也可以在nginx上配置location匹配jpg、png、html等结尾的所有请求转到静态服务池去。
- 还可以使用haproxy实现动静态分离，这个我也做过，修改haproxy的配置文件，设置mod为http七层代理，在frontent 里面加入acl访问控制，将所有html、jpg、png等结尾的访问定义为一个规则名，use_bakend 静态池 如果匹配的是这个

规则名的话，然后在后面定义backend静态池，同样地，加入静态服务器ip端口以及轮询方式等等，这样就实现了haproxy的动静分离了。

- **nginx做代理时的调度算法有哪些?**
- 有轮询、权重、iphash、urlhash等，但是在1.9版本后nginx开始支持基于tcp的四层负载均和反向代理，调度算法也和七层的有所不同，特别是一致性hash算法
- consistent_hash $remote_addr:跟iphash相似，可以根据用户ip来映射，这样做可以解决session问题
- consistent_hash $request_uri:跟urlhash相似，可以根据用户访问的url映射，这样做可以增加缓存的命中率，降低回源率
- consistent_hash $args:根据用户携带的参数来映射
- **nginx的upstream容错机制是怎么样的?**
- nginx默认判断失败节点状态以error错误和timeout超时状态为准，而不是以错误状态码来判断，因为毕竟有状态码返回，证明这个节点还是可以正常连接的。
- 我们可以添加proxy_next_upstream指令设置对错误状态码的请求转到其他后台处理，这样才会将这些状态码的请求视为失败节点状态。
- 如果所有的后台节点返回的都是错误状态码，那么就会返回给用户错误信息。
- 对单个后台节点来说，每返回一次错误状态码就会使这个节点的fails加一，当fails的值达到maxfails后，nginx就会将该节点置于down失效的状态，在接下来的failtimeout时间内不会再访问该节点，只有在这个时间过去后或者其他所有后台节点都失效了才会再次探测这个节点。
- 如果重新探测所有的节点还是失效，就会返回502网关错误，下次访问还会重复之前的步骤。
- **静态服务器的防盗链是怎么做的?**
- 我们之前用的是nginx静态服务器，nginx的防盗链是基于一个http_referes模块实现的。使用location匹配所有的图片，可以是jpg、png结尾的匹配，也可以是对一个目录的匹配，然后在下面加入一条valid_referes 代表白名单，只有写在后面的ip或主机才能访问到图片，其他的访问都将其重写到403forbidden去，防盗链就设置完毕了。当然，nginx有很多第三方模块，我们也可以使用第三方模块达到防盗的作用，我知道的一种就是ngx_http_accesskey模块，它可以通过md5加密来实现防盗链，但是需要程序的配合。
- **nginx和apache有什么优缺点，apache有几种工作模式，每个工作模式的优缺点。**
- nginx是一个轻量级服务，它比apache占用的内存等资源更少，它的抗并发能力比apache更强，因为它是异步非阻塞的，而apache是阻塞型的，nginx还可以用来

做反向代理和负载均衡，但是apache拥有的模块更多，功能更全面，apache还能支持动态页面，它的性能更稳定，而nginx相对bug较多。

- apache在Linux上有三种工作模式，分别是prework、worker和event。
- prefork是一个非线程型的、预派生的MPM（多进程处理模块），使用多个进程，每个进程在某个确定的时间只单独处理一个连接，效率高，也很安全，一个请求出现问题也不会影响到别的请求，但内存使用比较大，无法支持高并发请求。
- worker使用了多进程和多线程的混合模式，worker模式也同样会先预派生一些子进程，然后每个子进程创建一些线程，每个请求过来会被分配到一个线程来服务。它的内存使用相对减少，在高并发请求下表现也会更好。但是如果一个线程出现问题会导致同一个进程下的所有线程出现问题，而且在使用keepalive长连接的时候，某个线程会被一直占用，即使中间没有请求，也要等到超时才能释放，白白浪费资源。
- event模式是apache最新的工作模式，它和worker模式相似，只是在event工作模式中，会有一些专门的线程用来管理这些keep-alive类型的线程，当有真实请求过来的时候，将请求传递给服务器的线程，执行完毕后，又允许它释放，增强了在高并发场景下的请求处理。