

详细知识点跳转——> [面试知识点总结参考](#)

[TOC]

## 🌀自我介绍

简介
面试官您好，我的名字叫XXX，来自XX。我的专业是XXXXXXXXXXXXX。在XXXX年开始接触了Ubuntu及Centos等Linux系统，因此对Linux平台下的运维和相关运维开发工作有浓厚的兴趣。个人有较强的自学能力。我希望自己能够走向具有挑战性的职位，快速掌握新的运维技能，持续的学习实现自己的价值。XXXX年，我来到XXXXXXXXXXXXX公司工作，岗位是。其主要的工作内容是：“XX”。XXXXXXXXXXXXXXXXXXXX，XXXXXXXXXXXXXXXXXXXX。我的介绍完毕
【原离职公司】 XX
主要经营业务范围为XXXXXXXXXXXXXXXXXXXXXXXXXXXX服务

<b>职位描述</b>
1.负责公司MES日程系统运维管理工作,系统的运维监控,数据保证, 问题解决,故障处理和恢复
2.负责MES系统基准信息管理,使用者权限管理和维护, 工厂间生产基准信息设置和管理
3.负责将业务部门提出的需求转化为可实施方案, 后续方案实施及跟踪反馈
4.负责确保系统间数据的及时性和正确性,帮助客户提供必需数据, 及时解决客户数据异常岗位要求;
XXXXX【运维部门结构组成】：6人。4人项目技术, 其中一个是领导负责项目推进落地, 我属于其中的运维工作, 另外两个做项目C# (sharp吓普) 开发,
说一下你们公司怎么发版的 (代码怎么发布的) ?
回答：发布：jenkins配置好代码路径 (SVN或GIT) , 然后拉代码, 打tag。需要编译就编译, 编译之后推送到发布服务器 (jenkins里面可以调脚本) , 然后从分发服务器往下分发到业务服务器上。
辞职原因?
为了自己未来能够接触更具有挑战性的职位, 掌握新的运维技能实现自己的价值, 而上家公司的发展方向和个人成长不太相符, 不能满足我的发展, 晋升空间也比较狭窄, 因此我想换一个发展平台, 给自己找到更多的可能。
选择贵公司的原因：公司工作的氛围和企业文化比较契合自己。
公司的网站主体架构?
LNMP
个人对加班的看法?
加班能够使自己学习其他更多的专业知识和技能, 以弥补自己对公司业务方面的欠缺, 提升自己对公司的价值。
个人对运维的理解?
对范围内的硬件进行巡检工作, 及时发现异常硬件
确保各业务系统正确稳定运行, 掌握业务高峰状态
利用可行的监控手段监控各业务节点服务器的性能标准
面试结束时 我需要提问的问题 ?
1、我在该岗位可能会遇到最大的挑战是什么, 胜任这个岗位我还需要学习哪些技术知识?
2、如何考评自己在试用期内的的工作表现?

## ☞CentOS启动流程：

### 启动流程：

POST开机加电自检，加载BIOS的硬件信息，读取第一个启动设备MBR中的grub

通过grub读取内核，内核会尝试驱动所有硬件设备并挂载临时根目录

通过临时根目录加载核心模块，接着卸载根目录挂载真正根目录

内核初始化systemd取代init并进入开启流程，初始化系统

启动tty执行/bin/login程序（在终端上显示用户登录界面）

用户登录后将由shell控制主机

## 🔗Linux系统性能监控和管理命令

### linux系统中你会用到什么命令查看硬件使用状态信息？

回答：lscpu(查看cpu信息)、free -m（查看内存信息）、df -h（查看硬盘分区信息）、top（还可以动态查看cpu、内存使用情况的信息），/proc/目录下也可以查看很多硬件信息。

命令	作用
ps	ps命令以树状结构显示当前系统进程
ps	ps是linux系统中查看进程的有力工具之一。man帮助指明其用于报告当前系统所有进程的一个快照
nice	使用nice命令指定一个调度优先级来运行某程序
renice	renice命令可以更改一个正在运行的进程的优先级
pgrep	pgrep和pkill命令大部分选项相同，也就是大部分功能相同；但是pgrep一般用来基于进程名搜索某进程，
pkill	一般基于进程名来发送相关信号给某进程
kill	kill命令一般用来结束某进程，但是其还有其他功能，用于发送特定的信号给相关进程来控制某进程
killall	killall命令单纯的基于进程名来结束某进程
top	监控系统进程
htop	其使用不同的颜色来标识不同的信息，甚至支持鼠标点击相应的选项；来自EPEL源
free	查看内存空间使用情况
vmstat	命令查看虚拟内存信息
iostat	统计CPU和设备IO信息
iftop	显示带宽使用情况
pmap	显示某进程对应的内存映射
dstat	dstat命令用来统计系统资源(代替vmstat和iostat)
iotop	iotop命令用来监视磁盘I/O使用状况
nload	nload命令查看网络实时吞吐量

## 🔗tcpdump 抓包工具

### tcpdump介绍

tcpdump能够将网络中传送的数据包完全截获下来提供分析。通过将网络接口设置为混杂模式来嗅探网络数据包，抓取目标MAC地址为当前接口及广播MAC地址的数据包。通过expression关键字指明抓取的是什么类型、方向的数据包。

选项
-i interface
-w file
-nn: IP和端口数字显示
-A: ASCII码显示
-X: hex和ASCII显示
-XX: 包含链路层信息
-v: 详细信息 -vv: 更详细信息

tcpdump示例
tcpdump -i eth0 # 抓取eth0接口的数据包
tcpdump -i eth0 -nn # 第一个n表示将主机名显示为IP数字，第二个n表示将服务端口名显示为数字端口
tcpdump -i eth0 -nn port 80 # 抓取访问80端口的数据包

## 🔗 软硬链接

硬链接
创建硬链接会增加额外的记录项以引用文件，对应于同一文件系统上同一个物理文件 每个目录引用相同的inode号，创建时链接数递增
硬链接，以文件副本的形式存在。但不占用实际空间
无法跨文件系统创建、无法给目录或不存在的文件创建
删除其中一个硬链接文件不会影响其他有相同 inode 号的文件

符号（或软）链接
软链接链的数据内容是它所指向的路径
可以跨分区创建、可以对目录或不存在的文件创建
软链接的大小为其指向的路径字符数
删除或创建软链接不会影响目标文件inode的引用计数

## 🔗 目录权限

## 目录rwx权限

读权限：可列出目录之下的内容（如ls命令查看目录下的文件名）。

写权限：可在目录内创建、删除文件。注意，要删除文件，对文件本身无需有任何权限。

可执行权限：可访问目录中的文件。访问文件需要对该文件路径包含的所有父目录拥有执行权限。

## 🐧Linux的进程状态

### 进程类型

交互进程：由一个shell启动的进程。交互进程既可以在前台运行，也可以在后台运行。

批处理进程：这种进程和终端没有联系，是一个进程序列。

监控进程（也称守护进程）：Linux系统启动时启动的进程，并在后台运行。

### 进程状态

运行态：running

就绪态：ready

睡眠态：可中断睡眠态：interruptable

不可中断睡眠态：uninterruptable

停止态：stopped,暂停于内存，但不会被调度，除非手动启动

僵死态：zombie，结束进程，父进程结束前，子进程不关闭

### Linux进程优先级

系统优先级：数字越小，优先级越高

0-139：每个优先级有140个运行队列和过期队列

实时优先级: 99-0 值最大优先级最高

nice值：-20到19，对应系统优先级100-139

### export var 解读

变量var作为环境变量，将会被该shell的子进程继承。默认情况下POSIX兼容shell里的变量都只是shell自己用的，不是环境变量，也不会被子进程继承。export var=xxx是bash等支持的简写语法。var=xxxcmd是只为单条命令设置var环境变量用的语法。

## 🐧常见服务端端口

服务：端口号	服务：端口号	服务：端口号
HTTP: 80	NAGIOS: 5666	UDP: 138
FTP: 21	MEMCHACHED:11211	TCP: 139
DNS: 53	MYSQL: 3306	POSTFIX: 25
POP3:110	TOMCAT: 8080	IMAP: 143
SMTP: 25	NFS: 2049	zabbix: 10051
SSH: 22	TLENET: 23	SQUID: 3128
NGINX: 80	HTTPS:443	DHCP: 68

## 🔗 四层负载、七层负载的区别

### 常用的负载均衡软件

Nginx、LVS、HAProxy

### 区别

四层负载是基于IP+端口的负载均衡；通过发布三层的IP地址（VIP），然后加四层的端口号，来决定哪些流量需要做负载均衡，对需要处理的流量进行NAT处理，转发至后台服务器，并记录下这个TCP或者UDP的流量是由哪台服务器处理的，后续这个连接的所有流量都同样转发到同一台服务器处理。

七层负载是基于URL等应用层信息的负载均衡；在四层的基础上（没有四层是绝对不可能有七层的），再考虑应用层的特征。

## 🔗 DNS域名解析

### 递归查询

本机向本地域名服务器发出一次查询请求，如果本地域名服务器无法解析，自己会以DNS客户机的身份向其它域名服务器查询，直到得到最终的IP地址告诉本机

### 迭代查询

本地域名服务器向根域名服务器查询，根域名服务器告诉它下一步到哪里去查询，然后它再去查，每次它都是以客户机的身份去各个服务器查询

### 浏览器输入域名到显示的背后操作流程？

进行url解析（DNS域名解析——>先去查看本地的hosts 文件，查看是否存在与其域名所对应的ip 有则，直接根据此ip 进行请求，反之——>先去本地域名查询（找根服务器）——>找根服务器——>获得ip 然后进行查询）

查看本地是否有缓存——>有则使用缓存，反之请求

发起请求之后，服务器进行处理

然后返回数据，前端设置对检测相应状态的函数 onreadystatechange

然后前端判断ready.state ==4 ,status ==200

## 🔗CDN

### 什么是 CDN？

CDN是内容分发网络，通过将服务内容分发至全网加速节点，利用全球调度系统使用户能够就近获取，有效降低访问延迟，提升服务可用性，CDN 服务器会提前对静态内容进行预缓存，避免大量的请求回源，导致主站网络带宽被占用而导致数据无法更新，也可以将数据根据访问的热度不同而进行不同级别的缓存

## 🔗DHCP

### DHCP请求IP地址的过程

- 1、主机在网络上发送广播寻找DHCP服务器
- 2、DHCP服务器向主机发送单播数据包，包含IP地址、MAC地址、域名信息以及地址租期
- 3、主机发送广播，正式向服务器请求分配已提供的IP地址
- 4、DHCP服务器向主机发送单播，确认主机的请求

### DHCP实现流程

- 1、发现阶段，即DHCP客户机寻找DHCP服务器的阶段
- 2、提供阶段，即DHCP服务器提供IP地址的阶段
- 3、选择阶段，即DHCP客户机选择某台DHCP服务器提供的IP地址的阶段
- 4、确认阶段，即DHCP服务器确认所提供的IP地址的阶段
- 5、重新登录。以后DHCP客户机每次重新登录网络时，无需发送发现信息，而是直接发送包含前一次所分配的IP地址的请求信息
- 6、更新租约。DHCP服务器向DHCP客户机出租的IP地址一般都有一个租借期限，期满后DHCP服务器便会收回出租的IP地址。如果DHCP客户机要延长其IP租约，则必须更新其IP租约

## 🔗TCP/IP



## TCP/IP

TCP:是一种面向连接的、可靠的、基于字节流的传输层通信协议，在简化的计算机网络OSI模型中，它完成第四层传输层所指定的功能

用户数据报协议（UDP）是同一层内另一个重要的传输协议。

## TCP与UDP的区别。

TCP：面向连接，可靠的，速度慢，效率低。

UDP：无连接、不可靠、速度快、效率高。

## http和https

HTTP：超文本传输协议，用于传输 Internet 浏览器使用的普通文本、超文本、音频和视频等数据，端口号为 TCP 的 80。

HTTPS：安全超文本传输协议，基于 HTTP 开发，提供加密，可以确保消息的私有性和完整性，端口号为 443 工作区 子系统

## TCP/IP：四层模型。

①网络接口层：对应物理层和数据链路层。

②网络层

③传输层

④应用层：包括会话层、表示层、应用层。

## TCP/IP原理说一下？TCP有哪几个状态，分别是什么意思？

以tcp/ip协议为核心,分五层。tcp工作在第4层，主要有tcp和udp协议。其中tcp是可靠协议，udp是不可靠协议。tcp传输之前，需要建立连接，通过三次握手实现。

TCP三次握手状态：首先是closed状态，当发起连接后，进入Listen状态，当三次握手之后，进入EST状态。三次握手中还有一个临时状态:SYN\_SENT。SYN\_SENT 当应用程序发送ack之后，进入EST状态,如果没有发送，就关闭closed。

## 🔗三次握手

### 三次握手

TCP是一个面向连接的协议。无论哪一方向另一方发送数据之前，都必须先在双方之间建立一条连接。

1、请求端（通常称为客户）发送一个SYN段指明客户打算连接的服务器的端口，以及初始序号ISN。（这个SYN段为报文段1）。

2、服务器发回包含服务器的初始序号的SYN报文段（报文段2）作为应答。同时，将确认序号设置为客户的ISN加1以对客户的SYN报文段进行确认。

3、客户将确认序号设置为服务器的ISN加1以对服务器的SYN报文段进行确认（报文段3）。通过这三个报文后建立连接。这个过程称为三次握手

## 🌀四次挥手

### 四次挥手

建立一个连接需要三次握手，而终止一个连接要经过4次握手。这由TCP的半关闭（half-close）造成的。

当客户端没有东西要发送时就要释放连接的时候，客户端会发送一个FIN为1的没有数据的报文，进入FIN\_WAIT状态，服务器收到后会给客户端一个确认，这时客户端那边不再发送数据信息（但仍可接收信息）。

客户端收到服务器的确认后进入等待状态，等待服务器请求释放连接。服务器数据发送完成后就向客户端请求连接释放，客户端收到后回复一个确认信息，又要进入TIME\_WAIT状态（等待2MSL时间，最大报文生存时间）。服务器收到后关闭连接。

## 🌀zabbix

### 为什么要使用监控

- 1.对系统不间断实时监控
- 2.实时反馈系统当前状态
- 3.保证服务可靠性安全性
- 4.保证业务持续稳定运行

### 监控主要监控什么指标？

CPU、内存、网络、磁盘利用率，WEB，API，存储

### zabbix具有哪些监控功能

主机性能监控、网络设备监控、数据库监控等

支持多种报警机制

支持自动发现网络设备和服务器

支持分布式

自定义监控项

实时绘图功能

### zabbix监控系统运行基础流程：

agentd需要安装到被监控的主机上，它负责定期收集各项数据，并发送到zabbix server端，zabbix server将数据存储到数据库中，zabbix web根据数据在前端进行展现和绘图。

### agentd收集数据主动和被动模式：

主动：agent请求server获取主动的监控项列表，并主动将监控项内需要检测的数据提交给server/proxy

被动：server向agent请求获取监控项的数据，agent返回数据。

### zabbix 怎么开启自定义监控

- 1、写一个脚本用于获取待监控服务的一些状态信息。
- 2、在zabbix客户端的配置文件zabbix\_agentd.conf中添加上自定义的“UserParameter”，目的是方便zabbix调用我们上面写的那个脚本去获取待监控服务的信息。
- 3、在zabbix服务端使用zabbix\_get测试是否能够通过第二步定义的参数去获取zabbix客户端收集的数据。
- 4、在zabbix服务端的web界面中新建模板，同时第一步的脚本能够获得什么信息就添加上什么监控项，“键值”设置成前面配置的“UserParameter”的值。
- 5、数据显示图表，直接新建图形并选择上一步的监控项来生成动态图表即可。

### zabbix 监控了多少客户端 客户端是怎么进行批量安装的,根据实际公司台数回答。

- 1、使用命令生成密钥。
- 2、将公钥发送到所有安装zabbix客户端的主机。
- 3、安装 ansible 软件，（修改配置文件，将zabbix 客户机添加进组）。
- 4、创建一个安装zabbix客户端的剧本。
- 5、执行该剧本。
- 6、验证。

## MySQL原理

### MySQL增删改查

创建数据库：

#create database 数据库名 （字符集设置 default character set utf8）；

删除数据库：

#drop database 数据库名；（误删了可以参考mysql的数据恢复）

查看数据库：

#show databases;

查看某一数据库的详细信息（字符集）

#show create database 数据库名；

## MySQL主流存储引擎

InnoDB和MyISAM。最常用的是InnoDB，MySQL5.5版本之后，MySQL的默认内置存储引擎已经是InnoDB

## Mysql主从复制原理

- 1.master服务器将数据的改变都记录到二进制binlog日志中，只要master上的数据发生改变，则将其改变写入二进制日志；
2. slave服务器会在一定时间间隔内对master二进制日志进行探测其是否发生改变，如果发生改变，则开始一个I/O Thread请求master二进制事件
3. 同时主节点为每个I/O线程启动一个dump线程，用于向其发送二进制事件，并保存至从节点本地的中继日志中
4. 从节点将启动SQL线程从中继日志中读取二进制日志，在本地重放，使得其数据和主节点的保持一致

## Mysql主从同步原理

- 1) MYSQL主从同步是异步复制的过程，过程需要开启3线程，master上开启bin-log日志（记录数据库增、删除、修改、更新操作）
- 2) Slave开启I/O线程来请求master服务器，请求指定bin-log中position点之后的内容
- 3) Master端收到请求，Master端I/O线程响应请求，bin-log、position之后内容返给slave
- 4) Slave将收到的内容存入relay-log中继日志中，生成master.info（记录master ip、bin-log、position、用户名密码）
- 5) Slave端SQL实时监测relay-log日志有更新，解析更新的sql内容，解析成sql语句，再slave库中执行
- 6) 执行完毕之后，Slave端跟master端数据保持一致

## ☞LVS 调度算法

### Lvs的负载原理

LVS是Linux虚拟服务器的简称，利用LVS提供的负载均衡技术和linux操作系统可实现高性能、高可用的服务器集群，一般LVS都是位于整个集群系统的最前端，由一台或者多台负载调度器组成，分发给应用服务器。工作在4层（就是TCP/IP中的传输层）

LVS是基于IP负载均衡技术的IPVS模块来实现的，IPVS实现负载均衡机制有三种，分别是NAT、TUN和DR

【分布式】是通过缩短单个任务的执行时间来提高效率，【集群】是通过增加单位时间内执行的任务数量来提高效率。

## 静态调度算法

循环调度算法：该调度算法是 LVS 最简单的调度策略，其将每个传入的请求发送到其列表的下一个服务器。

加权轮询调度算法：该调度策略旨在处理不同的性能的服务器在接收请求和处理请求时的权重呵呵优先顺序。

目标地址哈希调度：目标地址散列调度算法也是针对目标IP地址的负载均衡，但它是一种静态映射算法，通过一个散列（Hash）函数将一个目标IP地址映射到一台服务器。

源地址散列调度算法正好与目标地址散列调度算法相反

## 动态调度算法

最少连接调度算法：该调度策略将网络连接调度给建立连接数量最少的 RS 服务器。

加权的最少连接调度算法：该调度策略允许为每个 RS 服务器分配一个性能权重值。

基于位置的最少连接调度算法：该调度策略专用于目的 IP 的负载均衡。

基于位置的带复制的最少连接调度算法：该调度策略也用于目的 IP 的负载均衡。

最短期望时延调度算法：该算法将网络连接调度分配给预期最小时延的服务器。

不排队算法：该调度策略在有空闲的服务器时将请求调度到该空闲服务器。

# 🔗Haproxy

## 定义

HAProxy 提供高可用性、负载均衡以及基于 TCP 和 HTTP 应用的代理，支持虚拟主机，它是免费、快速并且可靠的一种解决方案。HAProxy 特别适用于那些负载特大的 web 站点，这些站点通常又需要会话保持或七层处理。HAProxy 运行在当前的硬件上，完全可以支持数以万计的并发连接。并且它的运行模式使得它可以很简单安全的整合进您当前的架构中，同时可以保护你的 web 服务器不被暴露到网络上。

# 🔗tomcat

## 定义

Tomcat 本质上就是一款 servlet 容器，因此 catalina 才是 Tomcat 的核心，其他模块都是为 catalina 提供支撑的。比如:通过 coyote 模块提供链接通信, Jasper 模块提供JSP 引擎, Naming 提供 JNDI 服务, Juli 提供日志服务。

Tomcat 为了实现支持多种 I/O 模型和应用层协议，一个容器可以对接多个连接器。但是单独的连接器或者容器都不能对外提供服务，需要把它们组合起来才能够工作，组合后的整体叫做 Service 件。值得注意的是，Service 本并没有涉及到请求的处理，只是起到封装连接器和容器的作用。Tomcat 服务器可以配置多个 service，这样的设计处于灵活性考虑。这样可以通过在 Tomcat 配置多个 Service，通过不同的端口来访问同一台服务器上部署的不同应用。

## Nginx 介绍

Nginx 是一款自由的、开源的、高性能的 HTTP 服务器和反向代理服务器；同时也是一个 IMAP、POP3、SMTP 代理服务器。

nginx版本 >=1.9 可以支持四层和七层负载，版本 <1.9只支持七层；

## nginx特性

模块化设计，具有较好的扩展性

高可靠性

支持热部署：不停机更新配置文件，升级版本和更换日志文件

低内存消耗：10000 个 keep-alive 连接模式下的非活动连接，仅需 2.5M 内存

nginx 专注于高性能、高并发性和低内存使用。在 I/O 操作层面上，其支持 event-driven 事件驱动的 I/O，POSIX AIO1 异步 I/O mmap 内存映射和 sendfile 机制等等加速 I/O 的特性。这些特性都是其性能远胜apache 的原因。

## Nginx 与 web 服务相关的功能

主要功能

支持虚拟主机 (server)

支持 keep-alive 和管道连接(利用一个连接做多次请求)

访问日志(支持日志缓冲提高其性能)

url rewrite 支持 URL 重写

支持路径别名

基于 IP 及用户的访问控制

支持速率限制及并发数限制

重新配置和在线升级而无须中断客户的工作进程

## 常规多进程模型 多线程方式

一般的 WEB 服务器处理请求要么是以多进程的方式，要么是以多线程的方式。而apache 的 prefork 模型就是使用多个进程去处理请求；其 worker 模型就是多个进程和多个线程处理请求。

多进程模型：服务器每接收到一个客户端请求就有服务器的主进程生成一个子进程响应客户端，直到用户关闭连接，这样的优势是处理速度快，各子进程之间相互独立，但是如果访问过大会导致服务器资源耗尽而无法提供请求。

多线程方式：与多进程方式类似，但是每收到一个客户端请求会有服务进程派生出一个线程来为客户方进行交互，一个线程的开销远远小于一个进程，因此多线程方式在很大程度上减轻了 web 服务器对系统资源的要求，但是多线程也有自己的缺点，即当多个线程位于同一个进程内工作的时候，可以相互访问同样的内存地址空间，所以他们相互影响，另外，一旦主进程挂掉则所有子线程都不能工作了，IIS 服务器使用了多线程的方式，需要间隔一段时间就重启一次才能稳定。

其它的工作模式都是在这两种工作方式上延伸和改进的。

## nginx组织模型

Nginx 采用了多进程的组织模型，Nginx 工作时有一个 Master 进程和多个 Worker 进程共同完成所有的服务。Master 进程的工作偏向服务本身的管理，Worker 进程则负责处理用户请求。

### 主进程(Master process)的功能：

读取 Nginx 配置文件并验证其有效性和正确性(nginx -t)

建立、绑定和关闭 socket 连接

按照配置生成、管理和结束工作进程

接受外界指令，比如重启、升级及退出服务器等指令

不中断服务，实现平滑升级，重启服务并应用新的配置或者升级失败进行回滚处理

开启日志文件，获取文件描述符

编译和处理 perl 脚本

### 工作进程(Worker process)的功能：

接受处理客户的请求

将请求依次送入各个功能模块进行处理

IO 调用，获取响应数据

与后端服务器通信，接收后端服务器的处理结果

缓存数据，访问缓存索引，查询和调用缓存数据

发送请求结果，响应客户的请求

接收主程序指令，比如重启、升级和退出等

### nginx常用有几种调度算法并详细说明？

常用的有3种调度算法（轮询、ip hash、权重）。

轮询：upstream按照轮询（默认）方式进行负载，每个请求按时间顺序逐一分配到不同的后端服务器，如果后端服务器down掉，能自动剔除。

ip hash：每个请求按访问ip的hash结果分配，每个访客固定访问一个后端服务器

权重：指定轮询几率，权重和访问比率成正比，用于后端服务器性能不均的情况。

### nginx为什么性能高于apache？

Nginx在处理大量的用户请求和高并发场景下的性能高于apache的原因是使用了大量的异步事务处理机制，nginx的异步操作是通过模块化、事件通知、大量使用回调函数和调优计时器来实现的，尽可能的不阻塞

nginx 不会为每个连接派生一个进程或线程，在大多数情况下，内存使用是非常保守和高效的。如此nginx 也节约了 CPU 周期，因为不存在进程或线程的创建和销毁的开销。nginx大部分工作是检查网络相关和存储相关的状态，初始化新的连接，将它们添加到主运行循环中，然后异步处理，直到完成。请求处理完成后连接被释放并从运行循环中删除。通过谨慎的使用系统调用，如poll 等支持接口的精确实现，nginx 通常可以在极端工作负载下消耗较低的CPU 使用率。

由于nginx 的进程不随请求的数量变化而变化，所以在配置 nginx 时可以将工作进程和 CPU 的某个核心绑定来优化性能和增强稳定性。在不同的工作负载下可以规定工作进程的数量和处理器核心的比例关系来最大化利用 CPU 的处理性能。

nginx 服务启动工作时会在内存中运行多个进程。包括一个主进程和几个工作进程，还有一些特殊用途的进程，特别是缓存加载器和缓存管理器。nginx 所有进程主要使用共享内存机制进行进程间通信。主进程使用 root 用户运行。缓存加载器、缓存管理器和 Worker 进程使用非特权用户运行。

## 🔗apache

### 定义

Apache HTTP服务器是一个模块化的服务器，可以运行在几乎所有广泛使用的计算机平台上。其属于应用服务器。Apache支持模块多，性能稳定，Apache本身是静态解析，适合静态HTML、图片等，但可以通过扩展脚本、模块等支持动态页面等

### Apache特点

高度模块化：core + modules

DSO：Dynamic Shared Object 动态加/卸载

MPM：multi-processing module 多路处理模块



## apache工作模式

prefork: 多进程 I/O 模型；在 prefork 模式中，随着 httpd 服务的启动，系统会创建一个父进程，该进程负责创建和启动固定数量的子进程且不响应请求，子进程则侦听请求并在请求到达时提供服务。

worker: 在 worker 模式中，随着 httpd 服务的启动，系统会创建一个 httpd 主进程，该进程(父进程)负责创建子进程。每个子进程又会创建一个固定数量的服务线程(在 ThreadsPerChild 指令中指定)，以及一个监听器线程，监听请求并在请求到达时将其传递给服务线程进行处理。

event: 事件驱动模型，基于 worker 模型发展而来。它实现了混合的多进程多线程服务模型，httpd 服务会创建一个父进程，该父进程创建多个子进程，每个子进程创建固定数量的服务线程，以及一个监听线程(监听请求并在请求到达时将其传递给服务线程进行处理)。

## ☞Tomcat 、 Apache、 Nginx的区别及优缺点

### Apache与Tomcat的比较

相同点：

两者都是Apache组织开发的、免费的、且都有HTTP服务的功能

不同点：

Apache是专门用于提供HTTP服务以及相关配置的，而Tomcat是Apache组织在符合Java EE的JSP、Servlet标准下开发的一个JSP服务器。

Apache是一个Web服务器环境程序,启用它可以作为Web服务器使用,不过只支持静态网页。

Apache:侧重于HTTPServer , Tomcat:侧重于Servlet引擎

Apache是Web服务器，Tomcat是应用（Java）服务器，它只是一个Servlet容器，可以认为是Apache的扩展，但是可以独立于Apache运行。

<b>Nginx与Apache比较</b>
nginx相对于apache的优点：
轻量级，同样起web 服务，比apache占用更少的内存及资源
抗并发，nginx 处理请求是异步非阻塞的，而apache 则是阻塞型的，在高并发下nginx 能保持低资源低消耗高性能
高度模块化的设计，编写模块相对简单
提供负载均衡
apache 相对于nginx 的优点：
apache的 rewrite 比nginx 的强大
支持动态页面
支持的模块多，基本涵盖所有应用
性能稳定，而nginx相对bug较多
两者优缺点比较：
Nginx 配置简洁, Apache 复杂
Nginx 静态处理性能比 Apache 高 3倍以上
Apache 对 PHP 支持比较简单，Nginx 需要配合其他后端用
Apache 的组件比 Nginx 多
apache是同步多进程模型，一个连接对应一个进程；nginx是异步的，多个连接（万级别）可以对应一个进程
nginx处理静态文件好,耗费内存少
动态请求由apache去做，nginx只适合静态和反向
Nginx适合做前端服务器，负载性能很好
Nginx本身就是一个反向代理服务器，且支持负载均衡
<b>总结</b>
Nginx优点：负载均衡、反向代理、处理静态文件优势。nginx处理静态请求的速度高于apache；
Apache优点：相对于Tomcat服务器来说处理静态文件是它的优势，速度快。Apache是静态解析，适合静态HTML、图片等。
Tomcat：动态解析容器，处理动态请求，是编译JSP\Servlet的容器，Nginx有动态分离机制，静态请求直接就可以通过Nginx处理，动态请求才转发请求到后台交由Tomcat进行处理。
Apache在处理动态有优势，Nginx并发性比较好，CPU内存占用低，如果rewrite频繁，那还是Apache较适合。

## Ansible

### ansible 特性

模块化：调用特定的模块，完成特定任务

支持自定义模块，可使用任何编程语言写模块

基于 Python 语言实现

部署简单，基于 python 和 SSH(默认已安装)，agentless，无需代理不依赖 PKI（无需 ssl），去中心化部署

安全，基于 OpenSSH

幂等性：一个任务执行 1 遍和执行 n 遍效果一样，不因重复执行带来意外情况

支持 playbook 编排任务，YAML 格式，编排任务，支持丰富的数据结构较强大的多层解决方案 role

### Ansible相关配置文件

/etc/ansible/ansible.cfg 主配置文件，配置 ansible 工作特性

/etc/ansible/hosts 主机清单

/etc/ansible/roles/ 存放角色的目录

### ansible的作用

ansible 的主要功用在于批量主机操作

**ansible你用过它的哪些模块，ansible同时分发多台服务器的过程很慢（它是逐台分发的），你想过怎么解决吗？**

用过ansible的（copy file yum ping command shell）等模块；ansible默认只会创建5个进程,所以一次任务只能同时控制5台机器执行.那如果你有大量的机器需要控制,或者你希望减少进程数,那你可以采取异步执行.ansible的模块可以把task放进后台,然后轮询它.这使得在一定进程数下能让大量需要的机器同时运作起来.

## redis

### 什么是 redis ?

Redis 是一个基于内存的高性能 key-value 数据库

<b>redis的特点?</b>
速度快，每秒可以处理超过10万次读写操作
支持丰富数据类型，支持 string, list, set, sorted set, hash
支持事务
丰富的特性：可用于缓存，消息，按 key 设置过期时间，过期后将会自动删除

<b>redis适用场景?</b>
会话缓存
全页缓存
队列
计数器
发布、订阅

## 🐳 Docker

Docker 于 2013 年开源，其基于 go 语言开发，是一个开源的 PaaS 服务
--

<b>Docker 相比虚拟机而言：</b>
交付速度更快，资源消耗更低，Docker 采用客户端/服务端架构，使用远程 API 来管理和创建 Docker 容器，其可以轻松的创建一个轻量级的、可移植的、自给自足的容器，
docker 的三大理念是 build(构建)、ship(运输)、run(运行)
Docker 遵从 apache 2.0 协议，并通过 namespace 及 cgroup 等 linux 内核技术来提供容器的资源隔离与安全保障等，所以 Docker 容器在运行时不需要类似虚拟机的额外资源开销，因此可以大幅提高资源利用率,简单地说 Docker 是一种用了新颖方式实现的轻量级虚拟机.类似于 VM 但是在原理和应用上和 VM还是有一定差别的，并且docker 的专业叫法是应用容器

<b>Docker 架构</b>
Docker 为客户端-服务器架构。Docker 服务端也就是 Docker 的守护进程(docker daemon)，Docker 客户端通过 REST API 和守护进程通信，Docker daemon 负责创建、运行和分发容器。客户端和服务端可以运行于同一系统，也可以使用客户端连接远程服务进程。在同一系统时客户端和服务端通过 UNIX sockets 通信，不在同一系统则通过网络接口通信。

### Docker比传统虚拟化技术强在哪？

1. 资源利用率更高：一台物理机可以运行数百个容器，但是一般只能运行数十个虚拟机。
2. 开销更小：不需要启动单独的虚拟机占用硬件资源。
3. 启动速度更快：可以在数秒内完成启动。
4. 体积小：不像虚拟机使用 GB 为计量单位，容器的大小使用 MB 为单位。

容器技术的出现则减少了中间大量的运行和转换环节，节约了大量不必要的性能损耗和体积占用。

### docker和虚拟机的区别

虚拟机是在一台物理机器上，利用虚拟化技术，虚拟出来多个操作系统，每个操作系统之间是隔离的。Docker是开源的应用容器引擎，依然需要先电脑上安装操作系统，然后安装Docker容器的管理器，才可以。虚拟机是在硬件级别进行虚拟化，而Docker是在操作系统的层面虚拟化；虚拟机是通过模拟硬件搭建操作系统

### Dockerfile ?

Dockerfile 是一个用来构建镜像的文本文件，文本内容包含了一条条构建镜像所需的指令和说明。

Dockerfile 的指令每执行一次都会在 docker 上新建一层。因此过多无意义的层，会造成镜像膨胀过大。

### Docker 优势

快速部署: 在极短时间内可以部署成百上千个应用，可以快速交付产品到生产中。

高效虚拟化: 不需要额外的 hypervisor 程序支持，直接基于 linux 内核提供的功能实现应用虚拟化，相比虚拟机大幅提高性能和效率。

节省成本: 提高服务器利用率，降低 IT 支出。

简化配置: 将运行环境和应用打包保存至容器，使用时直接启动即可。

快速迁移和扩展: 容器可以跨平台运行在物理机、虚拟机、公有云等环境，良好的兼容性可以方便将应用从 A 宿主机迁移到 B 宿主机，甚至是不同的平台之间迁移。

### Docker 不足

隔离性: Docker 管理的容器提供进程级别的隔离，而不是虚拟机接近硬件级别的隔离。这样的隔离性下，内核的 BUG 会影响任何运行于其上的容器。

大量的容器管理: 在大量的容器部署和管理方面，Docker 有不小的挑战，但是 google 的 Kubernetes 可以帮忙。

</div>

</main>



- © 2020 GitHub, Inc.
- [Terms](#)
- [Privacy](#)
- [Security](#)
- [Status](#)
- [Help](#)

</ul>

<a aria-label="Homepage" title="GitHub" class="footer-octicon d-none d-lg-block mx-lg-4" href="https://github.com">

<svg height="24" class="octicon octicon-mark-github" viewBox="0 0 16 16" version="1.1" width="24" aria-hidden="true"><path fill-rule="evenodd" d="M8 0C3.58 0 0 3.58 0 8c0 3.54 2.29 6.53 5.47 7.59.4.07.55-.17.55-.38 0-.19-.01-.82-.01-1.49-2.01-3.7-2.53-.49-2.69-.94-.09-.23-.48-.94-.82-1.13-.28-.15-.68-.52-.01-.53.63-.01 1.08.58 1.23.82.72 1.21 1.87.87 2.33.66.07-.52.28-.87.51-1.07-1.78-.2-3.64-.89-3.64-3.95 0-.87.31-1.59.82-2.15-.08-.2-.36-1.02.08-2.12 0 0 .67-.21 2.2-.82.64-.18 1.32-.27 2-.27.68 0 1.36.09 2 .27 1.53-1.04 2.2-.82 2.2-.82.44 1.1.16 1.92.08 2.12.51.56.82 1.27.82 2.15 0 3.07-1.87 3.75-3.65 3.95.29.25.54.73.54 1.48 0 1.07-.01 1.93-.01 2.2 0 .21.15.46.55.38A8.013 8.013 0 016 8c0-4.42-3.58-8-8-8z"></path></svg>

- [Contact GitHub](#)
- [Pricing](#)
- [API](#)
- [Training](#)
- [Blog](#)
- [About](#)

  You can't perform that action at this time.

```
<script crossorigin="anonymous" async="async" integrity="sha512-
wcQmT2vhcc1FV0aaAJV/M+HqsJ2Gq/myv16F3gCVBxykazXTs+i5fvxncSXwyG1CSfcrqmLFw/R/bmFYzprX2A=="
type="application/javascript" id="js-conditional-compat" data-
src="https://github.githubassets.com/assets/compat-bootstrap-59c4264f.js"></script>
<script crossorigin="anonymous" integrity="sha512-
Qb7XHCWiafGt8U6FsJrxJqGCgszWjKK1zxv1+fAjIz0HdaYppsw1GLxEInNMN2d1MBAznZNSZL8wAvvhKxYx3A=="
type="application/javascript" src="https://github.githubassets.com/assets/environment-
bootstrap-41bed71d.js"></script>
<script crossorigin="anonymous" async="async" integrity="sha512-
327XCyoytNB1vvFres7NYMY/gSME67CreRw+pR2DhNKBfEb2sv0pe/XkZUSqYY1gGdDnaGIKUAb4oRZBLdKBfw=="
type="application/javascript" src="https://github.githubassets.com/assets/vendor-
df6ed70b.js"></script>
<script crossorigin="anonymous" async="async" integrity="sha512-
expQLpAhVAJMHk7Z0KcmLWkovvFHkwsEb7RddC/hfCt62wuVC8JAZUWiTeScadRGkJBFX6UxgmrE/uhIt1LvqA=="
type="application/javascript" src="https://github.githubassets.com/assets/frameworks-
797a502e.js"></script>

<script crossorigin="anonymous" async="async" integrity="sha512-
SSrX6F9r70jZSm3nckZCEPJ5EjxOLc2MBGMS4vs8vtD87p8BNrTkek3KFUuGPB08RRu7DmofRLOA/00y80Y2Qg=="
type="application/javascript" src="https://github.githubassets.com/assets/github-
bootstrap-492ad7e8.js"></script>
```