面试综合

实际运用问题

1.有客户说访问不到你们网站,但是你们自己测试一下自己的内网外网都是可以访问的,你会怎么排查解决客户的问题?

1. 与客户交流，询问状况，最好能直接远程上手
2. 排除客户本地网络问题
3. 排除DNS解析问题
4. 最后一种较为特殊，当服务器为南方移动而客户为北方联通时，时常会发生某个时段访问不了的情况，这种可以利用cdn技术解决

2.你会用什么方法来查看某个应用服务的流量使用情况?

1. ifconfig eth0
2. iftop（需要安装 iftop 软件包）
3. iptraf (需要安装 iptraf-ng 软件包)
4. sar -n DEV（需要安装 sysstat 软件包）
5. nethogs eth0（需要安装 nethogs 软件包）
6. 查看访问日志（利用 awk 统计资源大小）
7. 通过 zabbix 查看软件流量

3.说一说你们公司怎么发布版本的?

1. jenkins配置好代码路径
2. git拉代码,打tag
3. 最后推送到发布服务器上往下分发到业务服务器上.

4.linux系统中你会用什么命令查看硬件使用的状态信息?

- uptime、lscpu 查看 CPU
- free 查看内存
- lsblk、df、iostat 查看磁盘
- ifconfig、ip a s 查看网卡
- dmidecode 查看主板设备信息

5.比如开发想找你查看tomcat日志，但是catalia.out特别大，你不可能用vi打开去看，你会怎么查看？如果你用 grep -i"error" 过滤只是包含error的行，我想同时过滤error上面和下面的行如何实现?

grep -A 5 -i "error" catalia.out / grep -B 5 -i "error" catalia.out / grep -C 5 -i "error" catalia.out

6.你在工作的过程中，遇到过你映像最深的是什么故障问题，你又是如何解决?

1. 网站访问特别慢，经过查询，发现是开发的一个 bug，查询数据库创建了 80G 的临时表，修复 bug 即可
2. 网站被植入恶意代码，通过数据对比，找到注入点，通过备份的数据修复
3. 时间数据不同步（导致所有证书异常，网站无法访问）

7.在linux服务器上，不管是用rz -y(xshell上传)命令还是tftp工具上传，我把本地的一个文件上传到服务器完成后，服务器上还是什么都没有，这有可能是什么问题?

- 服务器磁盘满了
- 文件的格式被破坏了

- 文件是普通用户身份上传的，没有给权限
- 上传的文件超过了该目录的空间范围

8.在linux系统中，一般都会有swap内存，你觉得使用swap内存有什么好处，在什么情况下swap内存才会被使用？你觉得在生产环境中要不要用swap内存?

好处：可以弥补物理内存大小的不足；一定程度的提高反映速度；减少对物理内存的读取从而保护内存延长内存使用寿命

当系统的物理内存不够用的时候，就需要将物理内存中的一部分空间释放出来，以供当前运行的程序使用。那些被释放的空间可能来自一些很长时间没有什么操作的程序，这些被释放的空间被临时保存到swap空间中，等到那些程序要运行时，再从swap中恢复保存的数据到内存中。这样，系统总是在物理内存不够时，才进行swap交换

必要时使用，一般不用，虚拟内存和物理内存的性能差距过大，只能一定程度上缓解问题，不能解决问题

9.如果你们公司的网站访问很慢，你会如何排查?

1. 看监控各cpu内存,磁盘剩余空间等数据判断是否由硬件引起,解决方法更换故障硬件,加cpu或内存
2. 网络方面,看网卡进出量判断是否因为业务量的上涨导致带宽不足,或者网络延迟问题.解决办法,增加带宽
3. 系统方面:是否为服务器之间负载率没配比好.轮询时权重有问题,或者没有做一些配置文件和内核的优化.
4. 分析日志,看反应的人告诉是服务器的某个应用出问题了还是页面调度问题,打开阅览器按F12看哪一块过慢.或者看后端服务日志最直接,看看是不是逻辑代码的问题了，一行行审代码，找到耗时的地方改造，优化逻辑。
5. 查看流量(Zabbix,ifconfig,sar,ping 延迟... ...)
6. 系统负载(Zabbix,uptime,sar,top,ps,free 查看 CPU 和内存)
7. 日志（数据库日志-慢查询日志、web 服务器日志、ELK）
8. DNS 解析、ss 端口状态、并发量、本机时间（防止时间错误）
9. 浏览器 F12（开发者工具）

10.线上服务器如何把很大的数据拷贝到另外一台服务器上

- rsync、wget、切割成小文件再传输（tar 分卷压缩或 split）

11.如果遭遇了CC攻击该怎么办

通过网站日志判断哪个IP是CC攻击并做限制

所有防护服务都不可能做到100%防住CC攻击，哪怕防护能力高达99.9%，也会出现遗漏，这时候需要通过人工鉴定和筛选，通过网站日志判断哪个IP是CC攻击

(1) 下载服务器网站日志

只需要相应的web服务器开启日志功能，并查找对应打得存放路径即可下载每日的网站日志

(2) 分析CC攻击特征

实际上，通过分析网站日志，很容易区分哪个IP是CC攻击，因为CC攻击是通过程序来抓取网页，与普通用户的区别还是很大。

举例说明：

普通人访问一个网页，会连续抓取网页的HTML文件、CSS文件、JS文件和图片等一系列相关文件，而CC攻击仅仅只会抓取一个URL地址文件，不会抓取其他类型的文件，其User Agent也大部分和普通浏览者不同，这就可以在服务器上很容易分辨出哪些访问者是CC攻击。

(3) 找到CC攻击的IP怎么操作

既然可以判断出攻击者的IP，那么预防就简单了，只要批量将这些IP屏蔽，即可达到防御CC攻击的目的

12.传统运维架构如何搭建

- PXE 自动化部署；脚本、LNMP、负载均衡集群、高可用（Haproxy、Nginx）；MyCAT、MHA、PXC 数据库集群
- 存储层、服务层、负载层、高可用层、缓存层
- 核心是中间件用哪些, 监控用什么

13.维护的网页卡了怎么解决

1. 网速问题,可能是网络原因造成业务量不能上去
2. 调度器配置有问题, 或者负载比的机器性能没有设置好
3. 后端的消息队列没有做好(缓存机制)
4. 服务器被攻击了

- 打开图片卡和页面卡不一样的问题, 如果页面打开卡, 跟nginx负载有关, 或者某一台负载有问题

14.监控主要监控哪些项

zabbix监控nginx, 使用--with-http_stub_status_module模块

```
Active connections  Nginx正处理的活动链接数个数；重要
server              Nginx启动到现在共处理了多少个连接。
accepts             Nginx启动到现在共成功创建几次握手。
handled requests    Nginx总共处理了几次请求。
Reading             Nginx读取到客户端的 Header 信息数。
Writing             Nginx返回给客户端的 Header 信息数。
Waiting             Nginx已经处理完正在等候下一次请求指令的驻留链接，开启。
Keep-alive的情况下，Waiting这个值等于active-（reading + writing）。 请求丢失数=(握手数-连接数)可以看出,本次状态显示没有丢失请求。
```

zabbix监控MySQL, 自带监控模板

zabbix监控redis

监控范围：

监控指标	Key	公式
Redis 服务状态	redis[server.stat,{#PORT}]	无
Redis 实际使用的内存	redis[server.mem.rss,{#PORT}]	无
Redis 有效使用的内存	redis[server.mem,{#PORT}]	无
Redis 内存碎片率	redis[server.mem.debris.rate,{#PORT}]	redis[server.mem.rss,{#PORT}] / redis[server.mem,{#PORT}]
Redis 读命中次数	redis[server.hit,{#PORT}]	无
Redis 读未命中次数	redis[server.miss,{#PORT}]	无
Redis 读命中率	redis[server.hit.usage,{#PORT}]	redis[server.hit,{#PORT}] / (redis[server.hit,{#PORT}] + redis[server.miss,{#PORT}]) * 100
Redis 并发连接数	redis[server.connect.now,{#PORT}]	无
Redis 最大连接数	redis[server.connect.max,{#PORT}]	无
Redis 命令执行总次数	redis[server.cmd.total,{#PORT}]	无
Redis slave数量	redis[server.slave.num,{#PORT}]	无
Redis QPS	redis[server.qps,{#PORT}]	无
Redis Cluster 集群开关（1：启用 0：关闭）	redis[cluster.enabled,{#PORT}]	无
Redis Cluster 集群状态	redis[cluster.stat,{#PORT}]	无
Redis Cluster 集群节点数量	redis[cluster.nodes,{#PORT}]	无
Redis Cluster 数据槽分配数量	redis[cluster.slots,{#PORT}]	无

15.监控下cpu使用负载暴涨(进程占用率暴涨)

(1) 数据库:

- 业务类型,例如电商, 业务量起来了, 比如用户注册和用户下单
- 连接数超标之后, 导致FullGC和YGC增多报错, 占用计算力原来越多.
- 解决方法: 看日志, 一般运维解决不了

(2) web：

- 开发的页面bug, 表单上传, 没有限制大小, 上传一个几十M的. 例如:
- 成绩统计上传, 传很大的文件, 客户那边没有对应的报错,误以为系统正常, 后面上传的表单都会堆在一起. 运维就是重启服务, 告知开发

16.数据库并发连接数

- 学校业务并发最高50多. 游戏业务会高很多, 比如手游.
- 2核4Gmysql承载并发大约在200

17.数据库备份

在一周中业务量最小的时候完全备份, 剩余时间增量备份, 一般使用innodbbackupx

18.日常排过什么错

- nginx页面访问失败, 返回码500
- DDOS攻击
- 服务器被植入挖矿程序
- 数据库MySQL主从不同步等

- 软件包版本冲突

19.磁盘满了, 删除nginx日志之后, 空间还是满的

在Linux或者Unix系统中，通过rm或者文件管理器删除文件将会从文件系统的目录结构上解除链接(unlink).然而如果文件是被打开的（有一个进程正在使用），那么进程将仍然可以读取该文件，磁盘空间也一直被占用。

重启nginx服务，或者用 echo "" > /logs/nginx.log清空日志文件，而不是直接删除。

知识点问题

1.ansible你用过哪些模块,他同时部署多台的服务器时会很慢,这是为什么?怎么解决?

1. 模块：copy、shell、lineinfile、replace、ping、yum
2. 影响ansible性能的原因有两种，一种是执行上的资源浪费，需要对其进行优化; 另一种是进程和反馈机制造成的限制，可以用async和poll 机制缓解

ansible优化：

1. 关闭 gathering facts

   ansible-playbook 的第 1 个步骤总是执行 gather facts，不论你有没有在 playbook 设定这个 tasks。如果你不需要获取被控机器的 fact 数据的话，你可以关闭获取 fact 数据功能

```
---
- hosts: 172.16.64.240
  gather_facts: no
```

2. SSH PIPElinING

- SSH pipelining 是一个加速 Ansible 执行速度的方法，默认为关闭，为了兼容不同的 sudo 配置(主要是 requiretty 选项)。如果不使用 sudo，建议开启，打开此选项可以减少ansible执行没有传输时 ssh 在被控制机器上执行任务的连接数。
- 如果使用 sudo，必须关闭 requiretty 选项
- 通过修改/etc/ansible/ansible.cfg 文件可以开启 pipelining
- pipelining=True

3. ControlPersist

ControlPersist 特性需要高版本的 SSH 才支持，CentOS 6 默认不支持，使用需要自行升级openssh。ControlPersist 即持久化 socket ，一次验证，多次通信，并且只需要修改 ssh 客户端，即 Ansible 主机即可。

```
# cat ~/.ssh/config
ControlPersist 4h
```

开启了 ControlPersist 特性后，SSH 在建立了 sockets 之后，节省了每次验证和创建的时间。在网络状况不是特别理想，尤其是跨互联网的情况下，所带来的性能提升是非常可观的

以上内容为 ansible 调优

因为ansible默认最大进程数为5，且默认为同步执行，只有当上一条命令执行结束，才会开始执行下一条命令，导致长时间命令占用进程，大大降低了并发效率，这里可调整fork值来增加并发进程数

2.nginx你用到哪些模块, 在proxy模块中你配置哪些参数

- 模块：upstream，stream，stub_stats，ssl，gzip，ngx_http_proxy_module

(1) http_proxy

1. proxy_pass URL

- Context: location, if in location, limit_except
- proxy_pass转发代理请求
- 当 root 与 proxy_pass 同时存在是,proxy 优先级更高

```
# proxy_pass后面的路径不带uri时，其会将location的uri添加到结尾，传递给后端主机
# http://HOSTNAME/uri -> http://host/uri
location /uri/ {
    proxy_pass http://host[:port];
    }
# proxy_pass后面的路径是一个uri时，其会将location的uri替换为proxy_pass的uri
# http://HOSTNAME/uri/ -> http://host/new_uri/
location /uri/ {
    proxy_pass http://host/new_uri/;
    }
# 如果location定义其uri时使用了正则表达式的模式，则proxy_pass之后必须不能使用uri；用户请求时传递的uri将直接附加代理到的服务的之后
# http://HOSTNAME/uri/ -> http://host/uri/
location ~|~* /uri/ {
    proxy_pass http://host;
    }
# nginx uri 正则过滤中，~ 表示区分大小写，~* 表示不区分大小写
```

2. proxy_set_header field value

- 作用: 设定发往后端主机的请求报文的请求首部的值
- Context: http, server, location
- 前端的nginx代理,可以捕获客户端发送来的请求报文首部,并保存为$proxy_add_x_forwarded_for,此值可以传递给后续的代理服务器

```
proxy_set_header X-Real-IP  $remote_addr;
# 将请求的客户端远程地址传送给后端服务器
```

3. proxy_cache_path

- 定义可用于proxy功能的缓存
- Context: http

```
# proxy_cache_path path [levels=levels] [use_temp_path=on|off]
keys_zone=name:size [inactive=time] [max_size=size] [manager_files=number]
[manager_sleep=time] [manager_threshold=time] [loader_files=number]
[loader_sleep=time] [loader_threshold=time] [purger=on|off]
[purger_files=number] [purger_sleep=time] [purger_threshold=time];
```

4. proxy_cache zone | off

- 指明要调用的缓存，或关闭缓存机制
- Context: http, server, location

5. proxy_cache_key string

- 指明缓存中用于"键"的指定内容

- 默认值： proxy_cache_key $scheme$proxy_host$request_uri;
- 若希望公用缓存，则只是用$request_uri

6. proxy_cache_valid [code ...] time

- 定义对特定响应码的响应内容的缓存时长
- 若想全局生效,可以在server中定义,若希望局部uri生效,则在location中定义

```
# 定义在http{…}中
proxy_cache_path /var/cache/nginx/proxy_cache levels=1:1:1
keys_zone=pxycache:20m max_size=1g;
# 注意,缓存的目录需要事先创建
#定义在需要调用缓存功能的配置段，例如server{…}
proxy_cache pxycache;
proxy_cache_key $request_uri;
proxy_cache_valid 200 302 301 1h;
proxy_cache_valid any 1m;
```

7. proxy_cache_use_stale

- 指定缓存服务器与后端服务器无法通信时，何种情况下依旧使用过期的缓存内容来响应客户端

```
proxy_cache_use_stale error | timeout | invalid_header | updating | http_500
| http_502 | http_503 | http_504 | http_403 | http_404 | off …
```

8. proxy_cache_methods GET | HEAD | POST ...

- 定义允许使用缓存的请求方法：GET, HEAD, POST

9. proxy_hide_header field

- Context: http, server, location;
- 定义需要隐藏的响应报文首部
- 默认情况下，nginx不传递标题字段"Date"，"Server"，"X-Pad"，"X-Accel"...从代理服务器到客户端的响应。proxy_hide_header指令设置不被传递的附加字段。
- 一般nginx反向代理会配置很多站点，每个站点配置费时费力而少有遗漏，主机信息还是可能泄露。因此我们将proxy_hide_header配置在http区段，部分header信息无法用此方法，如关闭server信息，需要用此方式。

```
Syntax:server_tokens on | off | string;
Default:server_tokens on;
```

10. proxy_connect_timeout time;

- 代理服务器与后端服务器连接的超时时长
- 默认为60s，最大不超过75s

(2) http_headers

- nginx的 http_headers 模块允许在响应标头中添加"Expires"和"Cache-Control"头字段和任意字段。
- 模块功能：向由代理服务器响应给客户端的响应报文添加自定义首部，或修改指定首部的值。

1. add_header name value [always];

- 添加自定义首部

```
add_header X-Via  $server_addr;
# 添加代理服务器地址
```

2. expires [modified] time

```
expires epoch | max | off;
# 给出的日期/时间后，被响应认为是过时。如Expires:Thu, 02 Apr 2009 05:14:08
```

- GMT需和Last-Modified结合使用，用于控制请求文件的有效时间，当请求数据在有效期内时客户端浏览器从缓存请求数据而不是服务器端。当缓存中数据失效或过期，才决定从服务器更新数据。用于定义Expire或Cache-Control首部的值

3.我想查看access.log中哪个IP访问最多?

```
awk '{print $1}' access.log| sort | uniq -c |sort -rn -k 1 | head -1
awk '{ip[$1]++}END{for (i in ip) print i,ip[i]}' access.log | sort -rn -k 2 |
head -1
```

4.比如我访问百度网站，有什么方法可以跟踪经过了哪些网络节点?

- 用tracert(是路由跟踪实用程序，用于确定 IP 数据包访问目标所采取的路径)命令就可以跟踪,主要是查询本机到另一个主机的路由跳数和延迟情况

5.我需要查看某个时间段的日志(比如access.log日志)，如何实现?

```
awk -F "[/:]" '$7":"$8>="13:30" && $7":"$8<="14:30"'
/var/log/httpd/access_log
```

6.RAID

- RAID 磁盘阵列（Redundant Arrays of Inexpensive Disks：廉价冗余磁盘阵列）
- 过硬件/软件技术,将多个较小/低速的磁盘整合成一个大磁盘
- 阵列的价值:提升 I/O 效率（速度）硬件级别的数据冗余(备份)
- 附加优点：拥有更大容量的磁盘、比直接购买大容量/高速磁盘性价比更高
- RAID 实现方式：硬 RAID：由 RAID 控制卡管理阵列；软 RAID：由操作系统来管理阵列

7.如何写脚本删除过期日志?

```
#!/bin/bash
find /root/test/ -mtime +30 -name "*.log" | xargs -i mv {} /root/RecycleBin/;
chmod 777 /root/RecycleBin
rm -rf /root/RecycleBin/
```

8.python里怎么递归遍历文件

```
#使用listdir循环遍历
def getallfilesofwalk(dir):
    if not os.path.isdir(dir):
        print dir
        return
    dirlist = os.walk(dir)
    for root, dirs, files in dirlist:
        for file in files:
            print os.path.join(root, file)
```

9.TCP/IP原理说一下？TCP有哪几个状态，分别是什么意思？TCP与UDP的区别？

TCP原理：

- 以TCP/IP协议为核心,主要分为5层,tcp工作在第四层传输层,该层主要有tcp,udp协议.

状态：

- 有11个状态，可以分别概括为三次握手和四次断开。
  - 三次握手：客户端发起syn连接请求,服务端接受syn回给客户端syn和ack确认连接请求,客户端接收到ack后发送ack确认连接.
  - 四次断开：客户端发起FIN请求，服务端接受并返回ACK，此时服务端进入待关闭状态，向客户端发送FIN请求，客户端接受并返回ACK，连接随后关闭.

区别：

	TCP	**UDP**
优点	可靠，稳定。1.在数据传递之前，TCP会通过三次握手来建立连接2.在数据传递时，有确认、窗口、重传、拥塞控制机制3.数据传完后，会断开连接以节约系统资源	快，比TCP稍安全。1.UDP是一个无状态的传输协议，所以它在传递数据时非常快。2.UDP没有握手、确认、窗口、重传、拥塞控制等机制，较TCP而言被利用漏洞攻击的机会少一些，但这不代表UDP可以避免攻击，如UDP Flood攻击
缺点	慢，效率低，占用系统资源高，易被攻击。1.传递数据前，TCP需要先建立连接，这会消耗时间2.传递过程中，确认机制、重传机制、拥塞控制机制会消耗大量的时间，而且要在每台设备上维护所有的传输连接，每个连接都会占用硬件资源3.TCP的确认机制和三次握手机制使其容易被人利用，发起DOS、DDOS、CC等攻击	不可靠，不稳定。1.因为UDP没有可靠机制，在数据传递时，如果网络质量不好，会很容易丢包

区别归纳：

1. TCP面向连接，UDP面向无连接
2. TCP提供可靠的服务，UDP尽最大努力交付，不保证可靠交付
3. TCP面向字节流，UDP面向报文
4. 每一条TCP连接只能是点到点的，UDP支持一对一、一对多、多对一和多对多交互
5. TCP首部开销20字节，UDP只有8字节
6. TCP的逻辑通信信道是全双工的可靠信道，UDP则是不可靠信道

当网络通信质量有要求的时候，选择TCP，常用的有HTTP，HTTPS、FTP等文件传输协议，POP、SMTP等邮件传输协议。常见使用TCP协议的应用：浏览器(HTTP)、FlashFXP(FTP)、Outlook(POP、SMTP)、putty(Telnet、SSH)、QQ文件传输等

UDP一般用于可靠性要求不高的地方，如长视频，这种要求传输速率

10.nginx中rewrite有哪几个flag标志位?说一下意思

- last:当前的重写的规则;
- break:停止执行当前主机后续的重写指令;
- redirect返回302临时重定向,地址栏显示跳转后的地址;
- peimanent:返回301永久重定向,地址栏显示跳转后的地址

11.redis集群原理说一下，正常情况下mysql有多个库，redis也有多个库，我怎么进入redis集群中的第2个库？还有，我想查看以BOSS开头的值？redis持久化是如何实现（一种是RDS、一种是AOF），说一下他们有啥不一样？

redis集群原理：至少需要3主3从实现的一种无中心分布式存储结构,可以在多个节点间共享节点架构,出现单点故障时候,redis可以继续处理客户端请求

切库：单机情况下用select 2可以切换第2个库，select 1可以切换第1个库。但是集群环境下不支持select

持久化:就是把redis内存中的东西存入硬盘中主要用AOF,RBD

- RBD持久化分2种(SAVE,BGSAVE)
  - SAVE: 阻塞式持久化,把内存里的数据直接存在RDB(dump.rdb)文件下,直到文件创建完毕,期间redis不能做操作.
  - BGSAVE:非阻塞式持久化,在主进程内,新开一个子进程将redis内存中数据写入RDB文件中,同时可以接受客户端的请求.但是占用大量的内存使用率.
- AOF:全量写入持久化，与RDB的保存整个redis数据库状态不同，AOF的持久化是通过命令追加、文件写入和文件同步三个步骤实现的。AOF是通过保存对redis服务端的写命令（如set、sadd、rpush）来记录数据库状态的，即保存你对redis数据库的写操作。

12.Nginx(并发限制)

nginx通过limit_req模块和漏桶算法来限制IP并发连接数和请求数

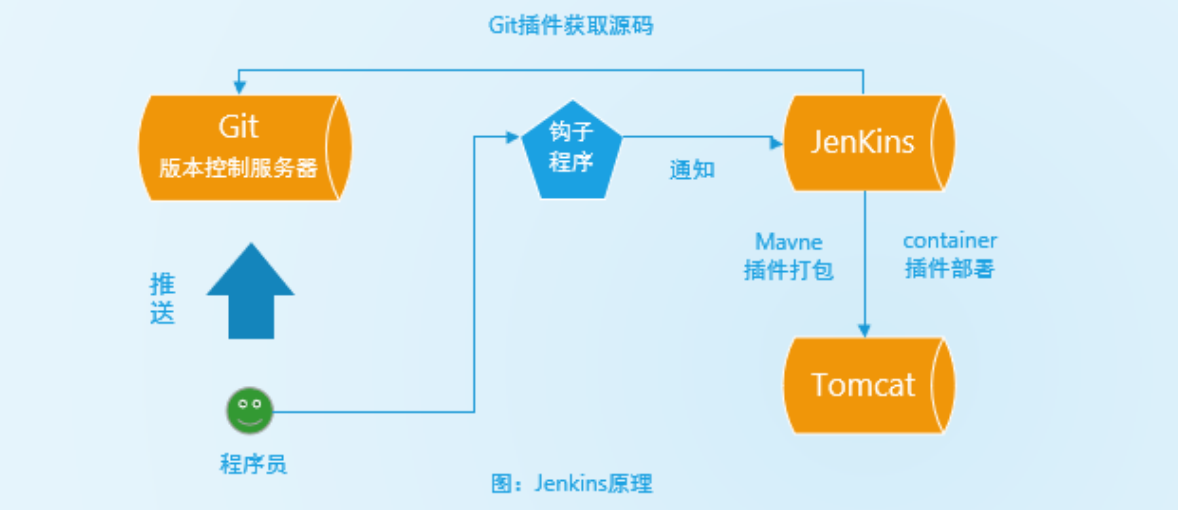
漏桶这个名字，其实就非常形象的描述了算法本身的原理。大家都知道，一个身上打了 n 个眼儿的桶，无论你倒进桶里的水多还是少，漏出来的水的流速也会保持稳定，这就是此算法的本质。再以 NGINX + PHP-FPM 为例，我们在 NGINX 配置里定义一个最大处理请求的速度，如果 PHP-FPM 的稳定处理速度峰值是 1000 RPS，那就在 NGINX 里定义处理请求速度最大为 1000 RPS。当 RPS 已经大于这个值的时候，多出来的请求就被 NGINX 这个桶暂时储存起来，排着队等待处理。在 NGINX 的精心照料下，PHP-FPM 会相对稳定的处理来自 NGINX 的请求，而不会出现突然暴增的请求让 PHP-FPM 处理不过来，甚至挂掉。

然而桶也有大小，NGINX 也一样，假如请求太多太多，桶都装不下了，那么桶将会把多出来的请求直接漏掉，返回 503 错误。

也可以返回错误429 Too Many Requests 用户在给定的时间内发送了太多的请求

13.Jekins工作流程

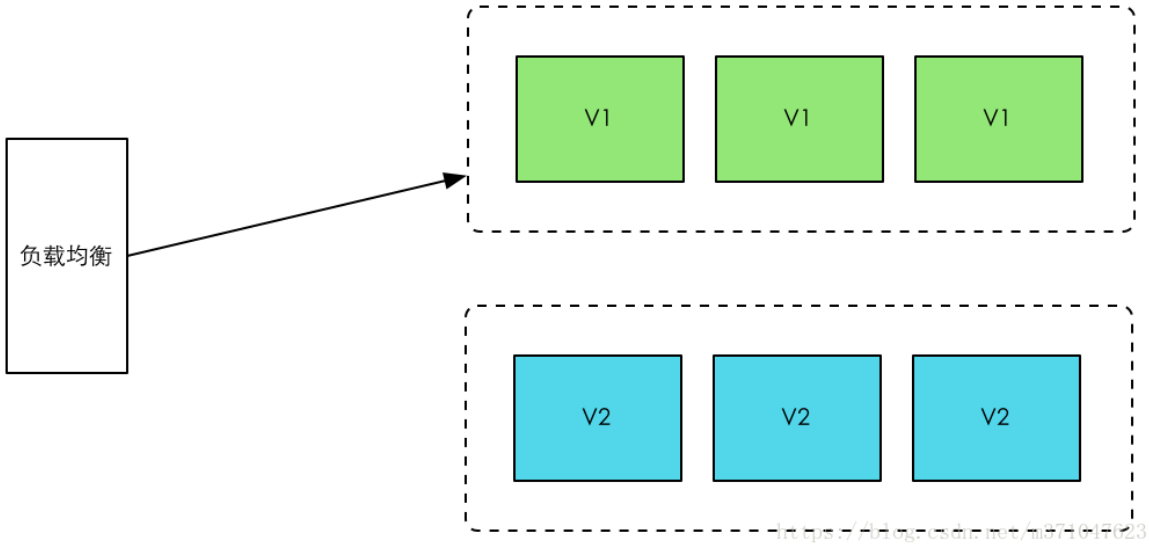
程序员提交代码到Git/SVN仓库，触发钩子程序向 JenKins 进行通知，Jenkins 调用Git/SVN插件获取源码，调用Maven打包为war包，调用Deploy to web container插件部署到Tomcat服务器



14.灰度发布

在一般情况下，升级服务器端应用，需要将应用源码或程序包上传到服务器，然后停止掉老版本服务，再启动新版本。但是这种简单的发布方式存在两个问题，一方面，在新版本升级过程中，服务是暂时中断的，另一方面，如果新版本有BUG，升级失败，回滚起来也非常麻烦，容易造成更长时间的服务不可用，为了解决这个问题，人们研究了多种方法，逐渐演变为灰度发布。

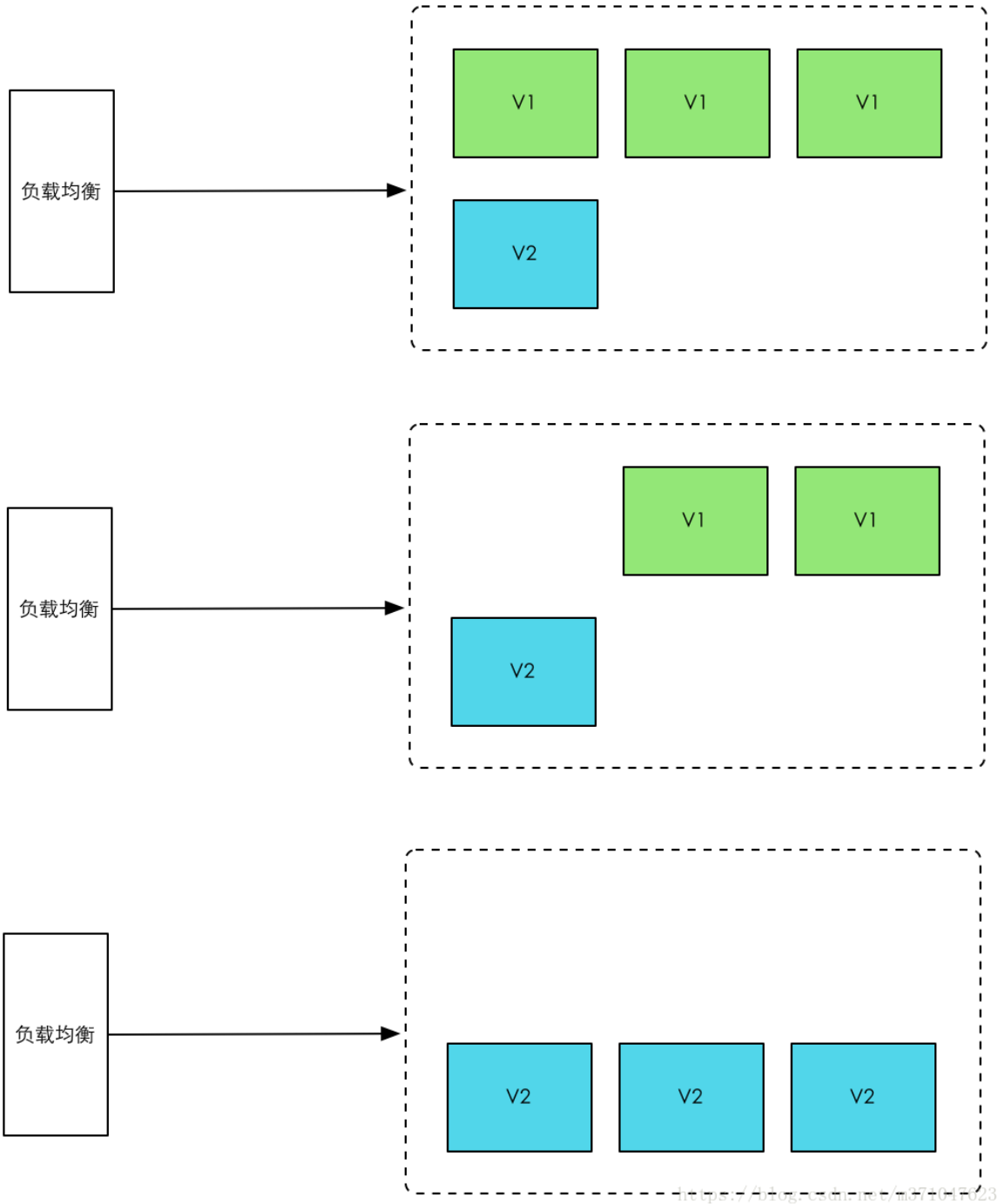
(1) 蓝绿部署



所谓蓝绿部署，是指同时运行两个版本的应用，如上图所示，蓝绿部署的时候，并不停止掉老版本，而是直接部署一套新版本，等新版本运行起来后，再将流量切换到新版本上。但是蓝绿部署要求在升级过程中，同时运行两套程序，对硬件的要求就是日常所需的二倍，比如日常运行时，需要10台服务器支撑业务，那么使用蓝绿部署，你就需要购置二十台服务器

(2) 滚动发布

滚动发布能够解决掉蓝绿部署时对硬件要求增倍的问题。



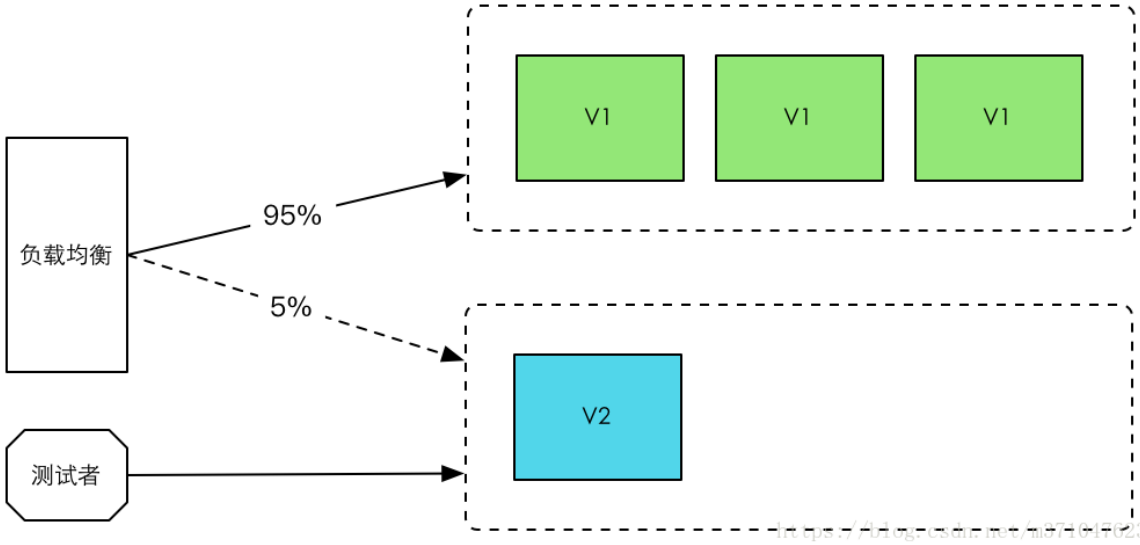
所谓滚动升级，就是在升级过程中，并不一下子启动所有新版本，是先启动一台新版本，再停止一台老版本，然后再启动一台新版本，再停止一台老版本，直到升级完成，这样的话，如果日常需要10台服务器，那么升级过程中也就只需要11台就行了

但是滚动升级有一个问题，在开始滚动升级后，流量会直接流向已经启动起来的新版本，但是这个时候，新版本是不一定可用的，比如需要进一步的测试才能确认。那么在滚动升级期间，整个系统就处于非常不稳定的状态，如果发现了问题，也比较难以确定是新版本还是老版本造成的问题。

为了解决这个问题，我们需要为滚动升级实现流量控制能力。

(3) 灰度发布

灰度发布也叫金丝雀发布，起源是，矿井工人发现，金丝雀对瓦斯气体很敏感，矿工会在下井之前，先放一只金丝雀到井中，如果金丝雀不叫了，就代表瓦斯浓度高。



在灰度发布开始后，先启动一个新版本应用，但是并不直接将流量切过来，而是测试人员对新版本进行线上测试，启动的这个新版本应用，就是我们的金丝雀。如果没有问题，那么可以将少量的用户流量导入到新版本上，然后再对新版本做运行状态观察，收集各种运行时数据，如果此时对新旧版本做各种数据对比，就是所谓的A/B测试

当确认新版本运行良好后，再逐步将更多的流量导入到新版本上，在此期间，还可以不断地调整新旧两个版本的运行的服务器副本数量，以使得新版本能够承受越来越大的流量压力。直到将100%的流量都切换到新版本上，最后关闭剩下的老版本服务，完成灰度发布

如果在灰度发布过程中（灰度期）发现了新版本有问题，就应该立即将流量切回老版本上，这样，就会将负面影响控制在最小范围内。

(4) 实现工具：脉冲云

面试注意项