1. Linux核心基础入门及系统进阶

1.1.1.1.1.1. 在Linux发行版本中,常见的块设备 (block devices)举例

硬盘，U盘，

字符设备、块设备主要区别是：在对字符设备发出读/写请求时，实际的硬件I/O一般就紧接着发生了，而块设备则不然，它利用一块系统内存作为缓冲区，当用户进程对设备请求能满足用户的要求时，就返回请求的数据，如果不能就调用请求函数来进行实际的I/O操作，因此，块设备主要是针对磁盘等慢速设备设计的，以免消耗过多的CPU时间来等待

1.1.1.1.1.2. 将/data/及其所有子文件全部授权给test用户,tester组,且单独对/data/file.sh添加执行权限,应该怎么操作?

```
chown -R test:tester /data/
chmod +x /data/file.sh
```

1.1.1.1.1.3. 请给出/tmp/目录默认权限是什么,为什么如此设置?

/tmp/的权限位为1777
使任何用户均可以向该目录写入文件,但仅允许修改该用户自己创建的文件,防止被非法修改.

1.1.1.1.1.4. 在bash中，export命令的作用是什么?

为其它应用程序设置环境变量

1.1.1.1.1.5. 有一个备份程序mybackup.sh，需要在周一至周五下午1点和晚上8点各运行一次，crontab应该如何写?

```
0 13,20 * * 1,2,3,4,5 /bin/bash /data/scripts/mybackup.sh  [此处请使用绝对路径]
```

1.1.1.1.1.6. 如何在不重启系统前提下,尝试挂载,新加入/etc/fstab中的文件系统?

```
mount -a
```

1.1.1.1.1.7. 在执行的脚本后面加入2>&1的作用是什么?它跟&>有什么差异?

2>&1的作用是将错误输出也输出到正确输出给定的目标,一般为 bash abc.sh > a.log 2&>1
&> 与 2>&1 功能完全一致,没有差异. 唯一差异是 在kernel2.4及以前不支持 &> 这种写法.

1.1.1.1.1.8. 使用mv命令移动一个文件都发生了什么?

如果在同一个文件系统内,则仅修改了inode中的元信息,重定向了一下文件指针.
如果在不同的文件系统之间移动,则会先在目标文件系统创建该文件,然后将原文件抽取数据流复制到目标位置,完成后再删除原文件的inode信息实现移动操作.

1.1.1.1.1.9. 在Linux中，提供TCP/IP包过滤功能的软件叫什么?

```
netfilter/iptables
```

1.1.1.1.1.10. 将当前目录下名为Protect的目录打包压缩的命令是什么?

```
tar zcvf Protect.tar.gz Protect
```

1.1.1.1.1.11. 在 bash shell 环境下，当一命令正在执行时，按下 control-Z 会产生什么效果?

```
将前台任务转入后台,可以使用fg唤醒
```

1.1.1.1.1.12. linux下通过mkdir命令创建一个新目录/data/path，它的硬链接数是多少，为什么?

```
答案：硬链接数是2个

Linux 文件系统最重要的特点之一是它的文件链接。链接是对文件的引用，这样您可以让文件在文件系统中多处被看到。不过，在 Linux 中，链接可以如同原始文件一样来对待。链接可以与普通的文件一样被执行、编辑和访问。对系统中的其他应用程序而言，链接就是它所对应的原始文件。当您通过链接对文件进行编辑时，您编辑的实际上是原始文件。链接不是副本。有两种类型的链接：硬链接和符号链接(软链接)。
硬链接只能引用同一文件系统中的文件。它引用的是文件在文件系统中的物理索引(也称为 inode)。当您移动或删除原始文件时，硬链接不会被破坏，因为它所引用的是文件的物理数据而不是文件在文件结构中的位置。硬链接的文件不需要用户有访问原始文件的权限，也不会显示原始文件的位置，这样有助于文件的安全。如果您删除的文件有相应的硬链接，那么这个文件依然会保留，直到所有对它的引用都被删除。
~]# ls -ial
   987372 drwxr-xr-x 2 root root  6 Aug 30 14:14 .
202150185 drwxr-xr-x 3 root root 18 Aug 30 14:14 ..
```

1.1.1.1.1.13. 你新建了一批用户，出于安全考虑，要求这些用户在第一次登录的时候 就必须要更改密码，怎么实现?

```
查看密码和账户过期信息：chage -l username
将密码设置为过期，用户登陆必须要更改密码：chage -d0 username 或 passwd -e username
```

1.1.1.1.1.14. 如何使文件只能写不能删除? 如何使文件不能被删除、重命名、设定链接接、写入、新增数据?

```
chattr +a 只能向文件中添加数据，而不能删除
chattr +i 文件不能被删除、改名、设定链接关系，同时不能写入或新增内容
```

1.1.1.1.1.15. 如何查看二进制文件的内容

```
一般通过hexdump命令 来查看二进制文件的内容。
hexdump -C XXX(文件名) -C是参数 不同的参数有不同的意义
-C 是比较规范的 十六进制和ASCII码显示
-c 是单字节字符显示
-b 单字节八进制显示
-o 是双字节八进制显示
-d 是双字节十进制显示
-x 是双字节十六进制显示
```

1.1.1.1.1.16. 符号链接与硬链接的区别

我们可以把符号链接，也就是软连接 当做是 windows系统里的 快捷方式，硬链接就好像是又复制了一份。

ln 3.txt 4.txt 这是硬链接，相当于复制，不可以跨分区，但修改3,4会跟着变，若删除3,4不受任何影响。
ln -s 3.txt 4.txt 这是软连接，相当于快捷方式。修改4,3也会跟着变，若删除3,4就坏掉了。不可以用了。

1.1.1.1.1.17. 保存当前磁盘分区的分区表

dd 命令是以个强大的命令，在复制的同时进行转换 dd if=/dev/sda of=./mbr.txt bs=1 count=512

1.1.1.1.1.18. 某/etc/fstab 文件中的某行如下： /dev/had5 /mnt/dosdata msdos defaults,usrquota 1 2 请解释其含义。

（1）第一列：将被加载的文件系统名；
（2）第二列：该文件系统的安装点；
（3）第三列：文件系统的类型；
（4）第四列：设置参数；
（5）第五列：供备份程序确定上次备份距现在的天数；
（6）第六列：在系统引导时检测文件系统的顺序。

1.1.1.1.1.19. 简述raid0 raid1 raid5 三种工作模式的工作原理及特点

RAID 0：连续以位或字节为单位分割数据，并行读/写于多个磁盘上，因此具有很高的数据传输率，但它没有数据冗余，因此并不能算是真正的RAID 结构。RAID 0 只是单纯地提高性能，并没有为数据的可靠性提供保证，而且其中的一个磁盘失效将影响到所有数据。因此，RAID 0 不能应用于数据安全性要求高的场合。
RAID 1：它是通过磁盘数据镜像实现数据冗余，在成对的独立磁盘上产生互为备份的数据。当原始数据繁忙时，可直接从镜像拷贝中读取数据，因此RAID 1 可以提高读取性能。RAID1 是磁盘阵列中单位成本最高的，但提供了很高的数据安全性和可用性。当一个磁盘失效时，系统可以自动切换到镜像磁盘上读写 ,而不需要重组失效的数据。简单来说就是：镜象结构，类似于备份模式，一个数据被复制到两块硬盘上。
RAID10:高可靠性与高效磁盘结构一个带区结构加一个镜象结构，因为两种结构各有优缺点，因此可以相互补充。主要用于容量不大，但要求速度和差错控制的数据库中。 RAID5：分布式奇偶校验的独立磁盘结构，它的奇偶校验码存在于所有磁盘上，任何一个硬盘损坏，都可以根据其它硬盘上的校验位来重建损坏的数据。支持一块盘掉线后仍然正常运行

1.1.1.1.1.20. 一个具有`-rwxrwxrwx`的文件具有什么权限？如果需要将其修改为`-rwxr-xr--`应该如何操作?

文件是777权限，对所有人均具有最大权限；
chmod 754 FILENAME

1.1.1.1.1.21. 检查硬盘 sda 读写速度的命令

dd if=/dev/zero bs=1024 count=1000000 of=/lGb

1.1.1.1.1.22. 检查系统I/O使用状态的命令

iostat -d /dev/sda1

1.1.1.1.1.23. 查询已安装的软件包，都生成了那些文件

```
rpm -ql sysstat
```

1.1.1.1.1.24. ps aux 中的 VSZ 代表什么意思，RSS 代表什么意思

```
VSZ:虚拟内存集,进程占用的虚拟内存空间
RSS:物理内存集,进程战用实际物理内存空间
```

1.1.1.1.1.25. 保存当前磁盘分区的分区表

```
# dd 命令是以个强大的命令，在复制的同时进行转换
d if=/dev/sda of=./mbr.txt bs=1 count=512
```

1.1.1.1.1.26. 修改内核参数

```
vi /etc/sysctl.conf  这里修改参数
sysctl -p  刷新后可用
```

2. Shell脚本编程与企业实战

2.1.1.1.1.1. 显示/test目录下的所有目录

```
ls -ld /test/*
```

2.1.1.1.1.2. 将文件/etc/a 下中除了 b文件外的所有文件压缩打包放到/home/a下，名字为a.gz

```
tar zcf /home/a/a.gz /etc/a/* --exclued=b
```

2.1.1.1.1.3. 计划每星期天早8点服务器定时发送一封内容为：test的邮件。发信人：user1@ab.com 收信人：test1@example.com，如何实现?

```
crontab -e
00 08 * * 7 echo "test" | /bin/mail -r user1@ab.com -s test test1@example.com
&>/dev/null
```

2.1.1.1.1.4. 编写个shell脚本将当前目录下大于100K的文件转移到/tmp目录下

```
find . -size +100K xargs -I {} mv {} /tmp
```

2.1.1.1.1.5. 如何把一个目录下的所有文件(不含目录)权限改为644?

```
find ./ ! -type d -exec chmod 644 \;
```

2.1.1.1.1.6. 写一个简单的shell脚本，脚本运行时让CTRL+C无法中断的该shell脚本?

```
在脚本里加入trap "" SIGINT 或 trap "" 2 以忽略SIGINT信号
```

2.1.1.1.1.7. find 找出找出当前目录下10天没有改变，大小大于4K的普通文件或目录

```
find . -type d -o -type f -size +4k -mtime +10
```

2.1.1.1.1.8. cp一个目录中的文件需要什么权限，mv呢？touch呢？rm呢？

```
cp一个目录中的文件，需要对这个目录有x权限，对这个文件有r权限
mv、touch、rm，都需要对这个目录有w和x权限，对文件权限没有要求
```

2.1.1.1.1.9. 找出access.log中访问top 10的ip地址

```
awk '{print $1}' nginx.log | grep -v "^$" | sort | uniq -c | sort -nr | head
```

2.1.1.1.1.10. shell下32位随机密码生成

```
cat /dev/urandom | head -1 | md5sum | head -c 32
```

2.1.1.1.1.11. 查找文件后缀是Log 的三天前的文件删除和三天内没修改过的文件

```
find / -name ".log" -mtime +3 -exec rm fr {} ; find /log ! -mtime -3
```

2.1.1.1.1.12. 如何让history 命令显示具体时间

```
HISTTIMEFORMAT="%Y-%m-%d %H:%M:%S "
```

2.1.1.1.1.13. 统计ip访问情况,要求分析nginx访问日志,找出访问页面数量在前十位的

```
awk '{print $1}' access.log |sort|uniq -c |head -n 10
```

2.1.1.1.1.14. 两台Linux服务器开放了987端口的SSH服务，使用user1用户登录1.1.1.1服务器，将/home/user1/data.tar.gz传至2.2.2.2服务器的/home/user2，并重命名为data2.tar.gz应如何操作?

```
scp -P 987 /home/user1/data.tar.gz user2@2.2.2.2:/home/user1/data2.tar.gz
```

2.1.1.1.1.15. 查找最后创建时间是3天前，后缀是*. log的文件

```
find /var/log -name **. log* -mtime +3
```

2.1.1.1.1.16. 如何调试shell脚本

```
方法1：#!/bin/bash -xv
方法2：在执行脚本时，bash -xv SHELLSCRIPTS.sh
```

2.1.1.1.1.17. 命令 “export” 有什么用？

```
使变量在子 shell 中可用.
```

2.1.1.1.1.18. & 和 && 有什么区别

```
& - 希望脚本在后台运行的时候使用它
&& - 当前一个脚本成功完成才执行后面的命令/脚本的时候使用它
```

2.1.1.1.1.19. 命令： name=John && echo ‘My name is $name’ 的输出是什么

```
My name is $name
```

2.1.1.1.1.20. ${variable:-10}与${variable: -10}有什么区别?

```
${variable:-10} - 如果之前没有给 variable 赋值则输出 10
${variable: -10} - 输出 variable 的最后 10 个字符
```

2.1.1.1.1.21. [[$string == abc*]] 与 [[$string == “abc*”]] 与什么区别

```
[[ $string == abc* ]] - 检查字符串是否以字母 abc 开头
[[ $string == “abc*” ]] - 检查字符串是否完全等于 abc*
```

2.1.1.1.1.22. 写出输出数字 0 到 100 中 3 的倍数(0 3 6 9 ...)的命令？

```
for i in {0..100..3}; do echo $i; done
```

2.1.1.1.1.23. [a == b]与 [a –eq b]有什么区别

```
[ a == b ] - 用于字符串比较
[ a –eq b ] - 用于数字比较
```

2.1.1.1.1.24. = 和 == 有什么区别

```
=  ：用于为变量赋值
== ：用于字符串比较
```

2.1.1.1.1.25. 如何列出以 ab 或 xy 开头的用户名？

```
egrep "^ab|^xy" /etc/passwd|cut -d: -f1
```

2.1.1.1.1.26. 如何打印数组的第一个元素？

```
echo ${array[0]}
```

2.1.1.1.1.27. 如何输出所有数组索引？

```
echo ${!array[@]}
```

2.1.1.1.1.28. 写脚本实现，在目录/tmp下找到100个以abc开头的文件，然后把这些文件的第一行保存到文件new中

```
答案1：
#!/bin/sh
for filename in `find /tmp -type f -name "abc*"|head -n 100`
do
sed -n '1p' $filename>>new
done

答案2：
find /tmp -type f -name “abc*” | head -n 100 | xargs head -q -n 1 >> new
```

2.1.1.1.1.29. 写脚本实现，把文件b中有的，但是文件a中没有的所有行，保存为文件c，并统计c的行数。

```
grep -vxFf a b | tee c | wc -l
```

2.1.1.1.1.30. 清除本机除了当前登陆用户以外的所有用户。

```
kill $(who -u|grep -v `whoami`|awk '{print $6}'|sort -u)
```

3. 操作系统内核及系统进程管理

3.1.1.1.1.1. 什么是进程

```
运行中的程序的一个副本，是被载入内存的一个指令集合，是资源分配的单位
```

3.1.1.1.1.2. 进程的基本状态

```
创建状态：进程在创建时需要申请一个空白PCB(process control block进程控制块)，向其中填写
控制和管理进程的信息，完成资源分配。如果创建工作无法完成，比如资源无法满足，就无法被调
度运行，把此时进程所处状态称为创建状态
就绪状态：进程已准备好，已分配到所需资源，只要分配到CPU就能够立即运行
执行状态：进程处于就绪状态被调度后，进程进入执行状态
阻塞状态：正在执行的进程由于某些事件（I/O请求，申请缓存区失败）而暂时无法运行，进程受
到阻塞。在满足请求时进入就绪状态等待系统调用
终止状态：进程结束，或出现错误，或被系统终止，进入终止状态。无法再执行
```

3.1.1.1.1.3. 找到未知进程的执行程序文件路径

```
ls -l /proc/$PID/exe
```

3.1.1.1.1.4. 如何查询系统负载

```
使用uptime命令
其中系统平均负载（1、5、15分钟的平均负载，一般不会超过1，超过5时建议警报）
```

3.1.1.1.1.5. 指定进程号，查看该进程打开的文件

```
lsof -p $PID 或 lsof -p $(pidof $COMMAND)
```

3.1.1.1.1.6. 让作业运行于后台(剥离终端)

```
nohup COMMAND &>/dev/null &
screen：COMMAND
tmux：COMMAND
```

3.1.1.1.1.7. 如何查看某进程所打开的所有文件

```
lsof -p 进程PID
```

3.1.1.1.1.8. 对于linux主机的cpu负载使用，什么情况下user的比例升高，什么情况下system的比例升高，请联系实际举例。

```
使用top命令可以查看cpu的负载使用
-us:用户进程消耗的CPU时间百分比 us的值比较高时，说明用户进程消耗的CPU资源多，如果长期超50%的使用，那么我们就该考虑优化程序算法或者进行加速（比如PHP/PERL）
-sy:内核进程消耗的CPU时间百分比（sy的值高时，说明系统内核消耗的CPU资源多，这并不是良性表现，我们应该检查原因）
-wa:IO等待消耗的CPU时间百分比 wa的值高时，说明IO等待比较严重，这可能由于磁盘大量随机访问造成，也有可能磁盘出现瓶颈（块操作）
-id:CPU处于空闲状态时间百分比,如果空闲时间(cpu id)持续为0并且系统时间(cpu sy)是用户时间的两倍(cpu us) 系统则面临着CPU资源的短缺
```

3.1.1.1.1.9. 在不umount的情况下，如何重新设置mount的参数。

```
mount -o remount,rw /
```

3.1.1.1.1.10. 新增一块存储设备，lvm操作的命令如何写

```
1、对新的存储设备分区并格式化为8e格式
2、创建物理卷（PV）
3、将PV加入到卷组（VG）
4、创建逻辑卷（LV）
5、格式化罗技卷并挂载
```

3.1.1.1.1.11. exec和source区别

```
所谓 source 就是让 script 在当前 shell 内执行、而不是产生一个 sub-shell 来执行。 exec 也是让 script 在同一个行程上执行，但是原有行程则被结束了。 他们的最大区别就是在于：原有行程会否终止。
```

3.1.1.1.1.12. 9.ps aux 中的 VSZ 代表什么意思， RSS 代表什么意思

```
VSZ:虚拟内存集,进程占用的虚拟内存空间
RSS:物理内存集,进程战用实际物理内存空间
```

3.1.1.1.1.13. 并行与并发

串行：一次只能取得一个任务并执行这一个任务
并行：可以同时通过多进程/多线程的方式取得多个任务，并以多进程或多线程的方式同时执行这些任务
并发：并发是一种现象：同时运行多个程序或多个任务需要被处理的现象；这些任务可能是并行执行的，也可能是串行执行的，和CPU核心数无关，是操作系统进程调度和CPU上下文切换达到的结果

3.1.1.1.1.14. 什么是上下文切换

在不断切换进程时，使CPU从一个进程切换到另一进程，需要保存进程A的运行环境及执行A到什么位置，以便下次切换回来继续执行。此时对前后两个进程的执行环境的切换叫做上下文切换。

3.1.1.1.1.15. 内核态与用户态

3.1.1.1.1.16. 解释一下进程和线程以及它们的区别

进程是具有一定功能的程序关于某个数据集合上的一次运行活动，进程是系统进行资源调度和分配的一个独立单位。
线程是进程的实体，是CPU调度和分派的基本单位，它是比进程更小的能独立运行的基本单位。
一个进程可以有多个线程，多个线程也可以并发执行

3.1.1.1.1.17. 进程的通信方式有哪些?

主要分为：管道、系统IPC（包括消息队列、信号量、共享存储）、SOCKET

3.1.1.1.1.18. 什么是缓冲区溢出？有什么危害？其原因是什么?

缓冲区溢出是指当计算机向缓冲区填充数据时超出了缓冲区本身的容量，溢出的数据覆盖在合法数据上。

危害有以下两点：

程序崩溃，导致拒绝额服务
跳转并且执行一段恶意代码
造成缓冲区溢出的主要原因是程序中没有仔细检查用户输入。

4. 网络通信与Linux系统网络安全

4.1.1.1.1.1. 获取eth0网卡上80端口的数据包信息

```
tcpdump -i eth0 port 80
```

4.1.1.1.1.2. 请用iptables控制来自192.168.1.2主机的80端口请求

```
iptables -A INPUT -s 192.168.1.2 -p tcp --dport 80 -j ACCEPT
```

4.1.1.1.1.3. TCP三次握手

(1)建立连接时，客户端发送SYN(SYN=j)包到服务器，并进入SYN_SEND状态，等待服务器响应、、确认
(2)服务器收到SYN包，必须确认客户端的SYN(ACK=j+1)，同时自己也发送一个SYN包，即SYN+ACK包此时服务器进入SYN_RECV状态
(3)客户端收到服务器的SYN+ACK包，向服务器发送确认包ACK(ack=k+1)，此包发送完毕客户端和服务器端进入ESTABLISHED状态，完成三次握手
为了保证服务端能收到客户端的信息并能做出正确的响应而进行前两次握手，为了保证客户端能够收到服务端的信息并能做出正确的响应而进行后两次响应

4.1.1.1.1.4. 四次挥手

1. 一端主动关闭连接。向另一端发送FIN包。
2. 接收到FIN包的另一端回应一个ACK数据包。
3. 另一端发送一个FIN包。
4. 接收到FIN包的原发送方发送ACK对它进行确认。

4.1.1.1.1.5. 如何将本地80端口的请求转发到8080端口,当前主机IP为192.168.16.1,其中本地网卡eth0

```
iptables -t nat -A PREROUTING -i eth0 -d 192.168.16.1 -p tcp --dport 80 -j
REDIRECT --to-ports 8080
```

4.1.1.1.1.6. 什么是NAT,常见分为那几种，DNAT与SNAT有什么不同，应用事例有那些?

NAT（Network Address Translation，网络地址转换）是将IP数据包头中的IP地址转换为另一个IP地址的过程。分为DNAT （目的网络地址转换）和SNAT（源网络地址转换）
SNAT主要是用于内网主机通过路由器或网关访问外网
DNAT将外部地址和端口的访问映射到内部地址和端口

4.1.1.1.1.7. 简述DDOS攻击的原理,有没有解决办法？有，如何解决?

分布式服务拒绝攻击就是用一台主服务器来控制N台肉鸡对目标服务器进行合理的资源请求，导致服务器资源耗尽而不能进行正常的服务。 几种流行的DDOS攻击方式：SYN/ACK FLOOD攻击、TCP全连接攻击、CC攻击(百科：攻击者借助代理服务器生成指向受害主机的合法请求，实现DDOS,和伪装就叫:cc(ChallengeCollapsar)。CC主要是用来攻击页面的。)
一个简单的测试： 首先是网站如果打不开的话，可以尝试着用3389连接一下服务器看看，然后还可以用PING命令来测试，再一种方式就是用telnet来登录80端口看看，看会不会出现黑屏。如果这些方式测试都连接不上的话，那就说明受到DDOS攻击了。 然后如果除了80端口之外的其他端口连接都正常，PING命令测试也正常，但就是80端口访问不了，然后看看IIS是否正常，可以把80端口改成其他端口测试，如果可以正常访问，那就说明很可能受到CC攻击。
防御DDOS攻击：
<1>要有充足的网络带宽和稳定安全的机房：选择口碑好、服务好、安全防护好点的机房，网络带宽直接决定了能抗受攻击的能力。
<2>软硬设备的防护：硬件DDOS防火墙黑洞、冰盾都不错，软件如web服务器都有相应的ddos防护模块，iptables,做单IP的并发限制，流量限制，syn及部分攻击限制。
<3>网站架构优化，避免单点提供服务，集群，冗余，负载均衡、缓存技术的架设。
<4>服务器系统自身的优化及安全参数调配
<5>采用高性能的网络设备

4.1.1.1.1.8. 出于安全考虑，如何实现让别人ping不通你的在线的服务器。

防火墙上用ACL封ICMP协议或者在服务器上使用iptables封icmp
在服务器上修改内核参数：echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_all

4.1.1.1.1.9. 给主机host：172.16.0.2增加gateway10.0.0.1

```
route add -host 172.16.0.2 gw 10.0.0.1
```

4.1.1.1.1.10. ftp、https、smtp、pops、ssh的端口号

ftp（20和21）、https(443)、smtp(25)、pops(110)、ssh(22)

4.1.1.1.1.11. 简述DNS 进行域名解析的过程

首先，客户端发出DNS 请求翻译IP 地址或主机名。DNS 服务器在收到客户机的请求后：
（1）检查DNS 服务器的缓存，若查到请求的地址或名字，即向客户机发出应答信息；
（2）若没有查到，则在数据库中查找，若查到请求的地址或名字，即向客户机发出应答信息；
（3）若没有查到，则将请求发给根域DNS 服务器，并依序从根域查找顶级域，由顶级查找二级域，二级域查找三级，直至找到要解析的地址或名字，即向客户机所在网络的DNS服务器发出应答信息，DNS 服务器收到应答后现在缓存中存储，然后，将解析结果发给客户机。
（4）若没有找到，则返回错误信息。

4.1.1.1.1.12. 什么是静态路由，其特点是什么？什么是动态路由，其特点是什么？

静态路由是由系统管理员设计与构建的路由表规定的路由。适用于网关数量有限的场合，且网络拓朴结构不经常变化的网络。其缺点是不能动态地适用网络状况的变化，当网络状况变化后必须由网络管理员修改路由表。
动态路由是由路由选择协议而动态构建的，路由协议之间通过交换各自所拥有的路由信息实时更新路由表的内容。动态路由可以自动 学习 网络的拓朴结构，并更新路由表。其缺点是路由广播更新信息将占据大量的网络带宽。

4.1.1.1.1.13. 用什么命令查询指定IP 地址的服务器端口

nmap 指定IP地址

4.1.1.1.1.14. 添加路由表并查看

```
route add -net 203.208.39.104 netmask 255.255.255.255 gw 192.168.1.1
netstat –r
```

4.1.1.1.1.15. 一台linux服务器，IP为192.168.0.199，仅开放TCP80端口，请在iptables上执行什么命令?

```
iptables -A INPUT -p tcp -dport 80 -d 192.168.0.199 -j ACCEPT iptables -A INPUT -p tcp -d 192.168.0.199 -j DROP
```

4.1.1.1.1.16. 限制 apache 每秒新建连接数为 1，峰值为 3

```
# 每秒新建连接数 一般都是由防火墙来做，apache 本身好像无法设置每秒新建连接数，只能设置最大连接：
iptables -A INPUT -d 172.16.100.1 -p tcp --dport 80 -m limit --limit 1/second  -j ACCEPT
```

4.1.1.1.1.17. TCP/UDP各有什么优缺点

TCP：优点：可靠、稳定
TCP的可靠体现在TCP在传输数据之前，会有三次握手来建立连接，而且在数据传递时，有确认、窗口、重传、拥塞控制机制，在数据传完之后，还会断开连接用来节约系统资源

TCP：缺点：慢，效率低，占用系统资源高，易被攻击
在传递数据之前要先建立连接，这会消耗时间，而且在数据传递时，确认机制、重传机制、拥塞机制等都会消耗大量时间，而且要在每台设备上维护所有的传输连接。然而，每个链接都会占用系统的CPU、内存等硬件资源。因为TCP有确认机制、三次握手机制，这些也导致TCP容易被利用，实现DOS、DDOS、CC等攻击

UDP：优点：快,比TCP稍安全
UDPm没有TCP拥有的各种机制，是一个无状态的传输协议，所以传递数据非常快，没有TCP的这些机制，被攻击利用的机制就少一些，但是也无法避免被攻击

UDP：缺点：不可靠，不稳定
因为没有TCP的那些机制，UDP在传输数据时，如果网络质量不好，就会很容易丢包，造成数据的缺失

适用场景：
TCP：当对网络通讯质量有要求时，比如HTTP、HTTPS、FTP等传输文件的协议， POP、SMTP等邮件传输的协议
UDP：对网络通讯质量要求不高时，要求网络通讯速度要快的场景

4.1.1.1.1.18. 常见HTTP状态码

200--请求已成功，请求所希望的响应头或数据体将随此响应返回
301--被请求的资源已永久移动到新位置
302--请求的资源临时从不同的 URI响应请求
401--当前请求需要用户验证
403--服务器已经理解请求，但是拒绝执行
404--请求的网页不存在
500--服务器内部错误
503--服务器暂时不可用

4.1.1.1.1.19. 对称加密与非对称加密

对称密钥加密，又称私钥加密，即信息的发送方和接收方用同一个密钥去加密和解密数据。

它的最大优势是加/解密速度快，适合于对大数据量进行加密，但密钥管理困难且较为不安全。
进行一对一的双向保密通信。
常见的对称加密算法：DES，AES等。

非对称密钥加密，又称公钥加密，它需要使用一对密钥来分别完成加密和解密操作，一个公开发布，即公开密钥，另一个由用户自己秘密保存，即私用密钥。信息发送者用公开密钥去加密，而信息接收者则用私用密钥去解密。

从功能角度而言非对称加密比对称加密功能强大且较为安全，但加密和解密速度却比对称密钥加密慢得多。
多对一的单向保密通信。
最常用的非对称加密算法：RSA

4.1.1.1.1.20. 什么是数字签名

数字签名必须保证实现以下三点功能：

报文鉴别。即接受者能够核实发送者对报文的签名。
报文的完整性。即接受者确信所收到的数据和发送者发送的完全一样而没有被篡改过。
不可否认。发送者事后不能抵赖对报文的签名。

4.1.1.1.1.21. cookie和session区别?

Cookie和Session都是客户端与服务器之间保持状态的解决方案，具体来说，cookie机制采用的是在客户端保持状态的方案，而session机制采用的是在服务器端保持状态的方案。

5. 常见网络服务及小型互联网架构

5.1.1.1.1.1. 如何实现Nginx代理的节点访问日志记录客户的IP而不是代理的IP?

1.首先要确保nginx配置文件这一行：proxy_set_header X-Real-IP $remote_addr;如果没有的话手动添加上。
2.编辑配置文件vim /etc/httpd/conf/httpd.conf 注，这是修改后的参数，将h%修改为%{X-Real-IP}i，

5.1.1.1.1.2. 简述网络文件系统NFS，并说明其作用。

网络文件系统是应用层的一种应用服务，它主要应用于Linux 和Linux 系统、Linux 和Unix系统之间的文件或目录的共享。对于用户而言可以通过 NFS 方便的访问远地的文件系统，使之成为本地文件系统的一部分。采用NFS 之后省去了登录的过程，方便了用户访问系统资源。

5.1.1.1.1.3. 请写出apache2.X 版本的两种工作模式，以及各自工作原理。如何查看apache 当前 所支持的模块，并且查看是工作在哪种模式下?

prefork(多进程，每个进程产生子进程)和worker（多线程，每个进程生成多个线程)
资源由 www.eimhe.com 美河学习在线收集分享
prefork 的工作原理是，控制进程在最初建立–StartServers 个子进程后，为了满足MinSpareServers 设置的需要创建一个进程，等待一秒钟，继续创建两个，再等待一秒钟，继续创建四个……如此按指数级增加创建的进程数，最多达到每秒32 个，直到满足MinSpareServers 设置的值为止。这就是预派生（prefork）的由来。这种模式可以不必在请求到来时再产生新的进程，从而减小了系统开销以增加性能。 worker 是2.0 版中全新的支持多线程和多进程混合模型的MPM。由于使用线程来处理，所以可以处理相对海量的请求，而系统资源的开销要小于基于进程的服务器。但是，worker 也使用了多进程，每个进程又生成多个线程，以获得基于进程服务器的稳定性。这种MPM 的工作方式将是Apache 2.0 的发展趋势。 可以通过命令httpd -l 可以查看apache 当前的模块，如果带有worker.c 就是工作在worker 模式下，如果有prefork.c 就是工作在prefork.c 的模式下。

5.1.1.1.1.4. ftp服务有几种模式，有什么区别?

分为主动模式与被动模式。主动为服务器发起21端口去访问客户端的随机端口，并通过服 务器的20端口来传数据•被动模式正好相反，由
客户端发起连接服务器的21端口，然后服 务器随机开启一个数据连接端口来传数据。

5.1.1.1.1.5. Apache两种工作模式的区别及优化?

prefork模式：这个多路处理模块(MPM)实现了一个非线程型的、预派生的web服务器，它的工作方式类似于Apache 1.3。它适合于没有线程安全库，需要避免线程兼容性问题的系统。它是要求将每个请求相互独立的情况下最好的MPM，这样若一个请求出现问题就不会影响到其他请求。

work模式：此多路处理模块(MPM)使网络服务器支持混合的多线程多进程。由于使用线程来处理请求，所以可以处理海量请求，而系统资源的开销小于基于进程的MPM。但是，它也使用了多进程，每个进程又有多个线程，以获得基于进程的MPM的稳定性

5.1.1.1.1.6. 什么是CGI

CGI：Common Gateway Interface 公共网关接口
当一个动态请求到达web服务器，会fork一个新的进程来运行外部程序处理这个请求，处理完毕后将数据返回给web服务器，之后服务器再将内容发还给用户，此时fork的进程也随之退出。如果还有新的动态请求，依然会fork出新的进程进程处理。
CGI 可以让一个客户端，从网页浏览器通过http服务器向执行在网络服务器上的程序传输数据；CGI描述了客户端和服务器程序之间传输的一种标准。

5.1.1.1.1.7. 什么是FastCGI

fastcgi的方式是，web服务器收到一个请求时，不会重新fork一个进程（因为这个进程在web服务器启动时就开启了，而且不会退出），web服务器直接把内容传递给这个进程（进程间通信，但fastcgi使用了别的方式，tcp方式通信），这个进程收到请求后进行处理，把结果返回给web服务器，最后自己接着等待下一个请求的到来，而不是退出

5.1.1.1.1.8. 请列举几个PHP常用设置

```
expose_php = On #响应报文显示首部字段x-powered-by: PHP/x.y.z，暴露php版本，建议为off
max_execution_time= 30 #最长执行时间30s
memory_limit=128M #生产不够，可调大
display_errors=off #调试使用，不要打开，否则可能暴露重要信息
display_startup_errors=off #建议关闭
post_max_size=8M #最大上传数据大小，生产可能调大，比下面项大
upload_max_filesize =2M #最大上传文件，生产可能要调大
max_file_uploads = 20 #同时上传最多文件数
date.timezone =Asia/Shanghai #指定时区
short_open_tag=on #开启短标签,如: <? phpinfo();?>
```

5.1.1.1.1.9. 如何使httpd与php进行结合

modules ：将php编译成为httpd的模块libphp5.so，只有prefork 模式才支持
FastCGI ：

5.1.1.1.1.10. 如何使php与mysql进行连接

1. 使用mysql扩展模块mysql.so连接数据（php7.x后淘汰）
2. 使用mysqli扩展连接数据库
3. 使用PDO(PHP Data Object)扩展连接数据库

5.1.1.1.1.11. 常见的第三方PHP加速工具有哪些?

```
APC (Alternative PHP Cache)
eAccelerator
XCache
Zend Optimizer和Zend Guard Loader
NuSphere PhpExpress
```

5.1.1.1.1.12. vsftp

5.1.1.1.1.13. nfs

5.1.1.1.1.14. rsync

5.1.1.1.1.15. phpmyadmin

6. 运维自动化部署与自动化实战

6.1.1.1.1.1. ansible有什么优点

```
部署简单，只需在主控端部署Ansible环境，被控端无需做任何操作；
默认使用SSH协议对设备进行管理；
幂等性  保证我们重复同一样操作的时候，得到的结果是一样的
通过Playbooks来定制强大的配置、状态管理；
支持API及自定义模块，可通过Python轻松扩展；
```

6.1.1.1.1.2. 什么是ansible模块

```
模块被认为是 Ansible 的工作单元。每个模块大多是独立的，可以用标准的脚本语言编写，如 Python、Perl、Ruby、bash 等。模块的一个重要属性是幂等性，意味着一个操作执行多次不会产生副作用。
```

6.1.1.1.1.3. 请列举几个常用的ansible模块

```
command ping yum copy service shell file replace user group
```

6.1.1.1.1.4. 什么是 Ansible 的 playbooks

```
Playbooks 是 Ansible 的配置、部署和编排语言，它是基于YAML语言编写的。他们可以描述您希望远程系统实施的策略，或者描述一般 IT 流程中的一系列步骤。
```

6.1.1.1.1.5. kickstart文件具有几个部分

```
命令段：指明各种安装前配置，如语言、键盘等
程序包段：指明要安装的软件包或包组
脚本段：安装前脚本及安装后脚本
```

6.1.1.1.1.6. 什么PXE

PXE：Preboot Excution Environment，预启动执行环境，是由Intel公司研发，基于Client/Server的网
络模式，支持远程主机通过网络从远端服务器下载映像，并由此支持通过网络启动操作系统，可以引导和安装Windows，linux等多种操作系统

6.1.1.1.1.7. 什么是cobbler

cobbler实质是PXE的二次封装，本身是一款Linux生态的自动化运维工具，基于Python2开发，用于自动化批量部署安装操作系统；

7. MySQL企业级实战及Mysql集群架构

7.1.1.1.1.1. 如何用MySQL 命令进行备份和恢复？以test 库为例，创建一个备份，并再用此备份进行恢复。

备份：mysqldump -u root -p test > test.sql
恢复：mysql < test.sql

7.1.1.1.1.2. 在mysql客户端查询工具中，如何获取当前的所有连接进程信息

```
show full processlist\G
```

7.1.1.1.1.3. 如何删除已满的数据库日志信息

在my.cnf中的[mysqld]段下面加入：expire-logs-days=7（设置自动清除7天前的logs），
重启mysql： 或者登录，mysql，执行：purge binary logs to 'mysql-bin.000003';
删除bin-log(删除mysql-bin.000003之前的而没有包含mysql-bin.000003)
如果是mysql主从环境的，删除日志，语句格式如下：
PURGE {MASTER | BINARY} LOGS TO ‘log_name’ PURGE {MASTER | BINARY} LOGS BEFORE ‘date’

7.1.1.1.1.4. 新安装MYSQL后怎样提升MYSQL的安全级别

一、 删除test数据库：drop database test; 删除不用的用户：drop user 'root'@'::1'; drop user ''@'centos6-2'; drop user ''@'locaohost'; 或全部删除，添加管理员：delete from mysql.user; grant all privileges on *.* to system@'localhost' identified by '123456' with grant option;
二、 对用户设置较复杂密码并严格指定对应账号的访问IP（可在mysql库中user表中指定用户的访问可访问IP地址，root限制为只允许本地登陆）
三、开启二进制查询日志和慢查询日志 四、mysql安装目录及数据目录权限控制：给mysql安装目录读取权限，给mysql日志和数据所在目录读取和写入权限 五、修改mysql默认端口，linux下可以通过iptables来限制访问mysql端口的IP地址

7.1.1.1.1.5. MYSQL的主从原理，怎么配置文件

A.master将改变记录到二进制日志(binary log)中（这些记录叫做二进制日志事件，binary log events）；
B.slave将master的binary log events拷贝到它的中继日志(relay log)；
C.slave重做中继日志中的事件，将改变反映它自己的数据。

（1）Slave上面的IO线程连接上Master，并请求从指定日志文件的指定位置（或者从最开始的日志）之后的日志内容；
（2）Master接收到来自Slave的IO线程的请求后，通过负责复制的IO线程根据请求信息读取指定日志指定位置之后的日志信息，返回给Slave端的IO线程。返回信息中除了日志所包含的信息之外，还包括本次返回的信息在Master端binary log文件的名称以及在Binary log中的位置；
（3）Slave的IO线程收到信息后，将接收到的日志内容依次写入到Slave端的RelayLog文件（mysql-relay-lin.xxxxx）的最末端，并将读取到的Master端的bin-log的文件名和位置记录到master-info文件中，以便在下一次读取的时候能够清楚的告诉master“我需要从某个bin-log的哪个位置开始往后的日志内容，请发给我”
（4）Slave的SQL线程检测到Relay Log中新增加了内容后，会马上解析该Log文件中的内容成为在Master端真实执行时候的那些可执行的查询或操作语句，并在自身执行那些查询或操作语句，这样，实际上就是在master端和Slave端执行了同样的查询或操作语句，所以两端的数据是完全一样的。

7.1.1.1.1.6. mysql主从复制的优点

<1> 如果主服务器出现问题， 可以快速切换到从服务器提供的服务；
<2> 可以在从服务器上执行查询操作， 降低主服务器的访问压力；
<3> 可以在从服务器上执行备份， 以避免备份期间影响主服务器的服务。

7.1.1.1.1.7. 什么是裸设备，他的好处是什么？，mysql支持裸设备吗?

裸设备：也叫裸分区（原始分区），是一种没有经过格式化，不被Unix/Linux通过文件系统来读取的特殊字符设备。裸设备可以绑定一个分区，也可以绑定一个磁盘。 好处：因为使用裸设备避免了再经过操作系统这一层，数据直接从Disk到数据库进行传输，所以使用裸设备对于读写频繁的数据库应用来说，可以极大地提高数据库系统的性能。当然，这是以磁盘的 I/O 非常大，磁盘I/O已经成为系统瓶颈的情况下才成立。如果磁盘读写确实非常频繁，以至于磁盘读写成为系统瓶颈的情况成立，那么采用裸设备确实可以大大提高性能，最大甚至可以提高至40%，非常明显。
mysql支持裸设备

7.1.1.1.1.8. socket和tcp访问mysql的区别?

TCP/IP 访问mysql：这种方式会在TCP/IP连接上建立一个基于网络的连接请求，一般是client连接跑在Server上的MySQL实例，2台机器通过一个TCP/IP 网络连接。（一般是mysql客户端跟服务端不在同一机器上） socket访问mysql：UNIX域套接字并不是网络协议，所以只能在MySQL客户端和数据库实例在同一台服务器上使用，用户可以在配置文件中指定套接字文件。

7.1.1.1.1.9. B树和B+树的区别

B树，每个节点都存储key和data，所有节点组成这棵树，并且叶子节点指针为nul，叶子结点不包含任何关键字信息。
B+树，所有的叶子结点中包含了全部关键字的信息，及指向含有这些关键字记录的指针，且叶子结点本身依关键字的大小自小而大的顺序链接
所有的非终端结点可以看成是索引部分，结点中仅含有其子树根结点中最大（或最小）关键字。 (而B 树的非终节点也包含需要查找的有效信息)

7.1.1.1.1.10. 如何授权test1用户从172.16.1.0/24访问数据库。

```
mysql> grant all on *.* to test1@'172.16.1.%' identified by '123456';
```

7.1.1.1.1.11. 什么是MySQL多实例，如何配置MySQL多实例?

在一台服务器上，mysql服务开启多个不同的端口，运行多个服务进程，这些mysql服务进程通过不同的socket来监听不同的数据端口，进而互不干涉的提供各自的服务。

7.1.1.1.1.12. delete和truncate删除数据的区别?

```
truncate table test执行更快，清空物理文件，清空表中的所有内容
delete from test是逻辑删除，按行删除，而且可以通过where语句选择要删除的行
```

7.1.1.1.1.13. MySQL Sleep线程过多如何解决?

```
mysql> show processlist\G
# mysqladmin -uroot -p123456 processlist
修改my.cnf文件里的wait_timeout的值，让其更小一些，默认wait_timeout =28800，这里改为100
mysql> set global wait_timeout=100;
mysql> show global variables like "wait_timeout";
```

7.1.1.1.1.14. sort_buffer_size参数作用? 如何在线修改生效?

```
mysql执行排序使用的缓冲大小。如果想要增加order by的速度，首先看是否可以让mysql使用索引而不是额外的排序阶段，如果不能，可以尝试增加sort_buffer_size变量的大小。
mysql> set global sort_buffer_size =131072;     #单位为B，即128KB，默认64K
```

7.1.1.1.1.15. 如何在线正确清理MySQL binlog?

```
自动清除
mysql> set global expire_logs_days=30;   #设置binlog过期时间为30天
手动清除
mysql> purge binary logs to "mysql-bin.000007";    #/删除mysql-bin.000007之前的所有binlog日志
```

7.1.1.1.1.16. 误操作执行了一个drop库SQL语句，如何完整恢复?

```
如果条件允许，操作前最好禁止外面一切服务器访问mysql数据库，这里假设禁止外面访问数据库，具体步骤如下：
1 手动切割binlog日志并记好切割好的binlog日志文件位置，这里假设为009，备份全部binlog日志
2 找到之前全备数据最后备份到的binlog文件位置并记好位置，这几假设为005
3 用mysqladmin命令将005到008binlog文件中的SQL语句分离出来，并找到drop库的语句将其删掉
4 将之前全备数据导入mysql服务器
5 将步骤3中分离出的SQL语句导入mysql服务器
6 将009binlog文件删除，再次刷新binlog日志，到此数据库已恢复成功。
```

7.1.1.1.1.17. 详述MySQL主从复制原理及配置主从的步骤

1 主：binlog线程，记录所有改变了数据库数据的语句，放进master上的binlog中
2 从：IO线程，在使用start slave之后，负责从master上拉取binlog内容，放进自己的relay log中
3 从：SQL执行线程，执行relay log中的语句。

配置步骤：
1 主库开启binlog日志功能
2 全备数据库，记录好binlog文件和相应的位置
3 从库上配置和主库的连接信息
4 将全备数据导入从库
5 从库启动slave
6 在从库上查看同步状态，确认是否同步成功

7.1.1.1.1.18. MySQL出现复制延迟有哪些原因？如何解决？

1 一个主库的从库太多，导致复制延迟
建议从库数量3-5个为宜，要复制的从节点数量过多，会导致复制延迟
2 从库硬件比主库差，导致复制延迟
查看master和slave的系统配置，可能会因为机器配置问题，包括磁盘IO、CPU、内存等各方面因素造成复制的延迟，一般发生在高并发大数据量的写入场景。
3 慢SQL语句过多
假如一条SQL语句执行时间是20秒，那么执行完毕到从库上能查到数据也至少是20秒，可以修改后分多次写入，通过查看慢查询日志或show full processlist命令找出执行时间长的查询语句或者大的事务。
4 从复制设计问题
主从复制单线程，如果主库写并发太大，来不及传送到从库就会导致延迟。更高版本的mysql可以支持多线程复制，门户网站则会自己开发多线程同步功能。
5 主从库之间网络延迟
主从库的网卡，网线，连接的交换机等网络设备都可能成为复制的瓶颈，导致复制延迟，另外，跨公网主从复制很容易导致主从复制延迟。
6 主库读写压力大，导致复制延迟
主库硬件要搞好一点，架构的前端要加buffer。

7.1.1.1.1.19. MySQL高可用方案有哪些，各自特点，企业如何选择？

1.　　主从复制+读写分离
优点：成本低、架构简单、易实施、维护方便
缺点：master出现问题后不能自动到slave上，需要人工干涉。
2.　　MySQL Cluster
优点：安全性高，稳定性高。可以在线增加节点
缺点：架构复杂，至少三个节点，对于引擎只能用ndb，不支持外键，管理复杂，部署费时而且是收费的。
3.　　Heartbeat /keepalived+双主从复制
优点：安全性、稳定性高，出现故障系统将自动切换，从而保证服务的连续性。
缺点：可能会发生脑裂
4.　　HeartBeat+DRBD+MySQL
优点：安全性、稳定性、出现故障系统将自动切换，从而保证服务的连续性。
缺点：只用一台服务器提供服务，成本高，可能发生脑裂

7.1.1.1.1.20. xtranbackup的备份、增量备份及恢复的工作原理？

XtraBackup基于InnoDB的crash-recovery功能，它会复制InnoDB的data file，由于不锁表，复制出来的数据是不一致的，在恢复的时候使用crash-recovery，使得数据恢复一致。
InnoDB维护了一个redo log，又称为transaction log（事务日志），它包含了InnoDB数据的所有改动情况。当InnoDB启动的时候，它会先去检查data file和transaction log，并且会做两步操作：
XtraBackup在备份的时候，一页一页的复制InnoDB的数据，而且不锁定表，与此同时，XtraBackup还有另外一个线程监视着transaction log，一旦log发生变化，就把变化过的log pages复制走。为什么要着急复制走呢？因为transaction log文件大小有限，写满之后，就会从头再开始写，所以新数据可能会覆盖到旧的数据。
在prepare过程中，XtraBackup使用复制到的transaction log对备份出来的InnoDB data file进行crash recovery

7.1.1.1.1.21. mysql的权限怎么管理?

只给insert,update，select和delete四个权限即可。对权限管理比较严格的情况delete也不会给，让研发走逻辑删除位标识。

7.1.1.1.1.22. 请详细描述char(4)和varchar(4)的差别

char(4)定义的是固定长度4，存储时，如果字符数不够4位，会在后面用空格补全存入数据库。
varchar(4)定义的是变长长度，存储时，如果字符没有达到定义的位数4时，也不会在后面补空格。

7.1.1.1.1.23. 如何监控主从复制是否故障?

查看slave端的IO和SQL进程状态是否OK，同步延迟时间是否小于1分钟

```
mysql> show slave status\G
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
Seconds_Behind_Master: 0
```

7.1.1.1.1.24. MySQL数据库如何实现读写分离?

1. 通过程序实现读写分离（性能，效率最佳，推荐
PHP和Java程序都可以通过设置多个连接文件轻松的实现对数据库的读写分离，即当select时，就去连接读库的连接文件，当update、insert、delete是就去连接写库的连接文件。
2. 通过软件实现读写分离
MySQL-proxy,Amoeba等代理软件也可以实现读写分离功能，但最常用最好用的还是程序实现读写分离。
3. 开发dbproxy

7.1.1.1.1.25. 生产一主多从从库宕机，如何手工恢复?

处理方法：重做slave
1. 停止slave
2. 导入备份数据
3. 配置master.info信息
4. 启动slave
5. 检查从库状态

7.1.1.1.1.26. MySQL的SQL语句如何优化?

1. 在表中建立索引，优先考虑where、group by使用到的字段
2. 尽量避免使用select *，返回无用的字段会降低查询效率
3. 尽量避免使用in和not in，会导致数据库引擎放弃索引进行全表扫描
4. 尽量避免使用or，会导致数据库引擎放弃索引进行全表扫描
5. 尽量避免在字段开头模糊查询，会导致数据库引擎放弃索引进行全表扫描

7.1.1.1.1.27. 如果发现CPU，或者IO压力很大，怎么定位问题?

1、首先我会用top命令和iostat命令，定位是什么进程在占用cpu和磁盘io；
2、如果是mysql的问题，我会登录到数据库，通过show full processlist命令，看现在数据库在执行什么sql语句，是否有语句长时间执行使数据库卡住；
3、执行show engine innodb status\G命令，查看数据库是否有锁资源争用；
4、查看mysql慢查询日志，看是否有慢sql；
5、找到引起数据库占用资源高的语句，进行优化，该建索引的建索引，索引不合适的删索引，或者根据情况kill掉耗费资源的sql语句等

8. 基于Nginx的web网站架构LNMP及文件存储

8.1.1.1.1.1. 什么是Nginx?

Nginx是一个web服务器和反向代理服务器，用于HTTP、HTTPS、SMTP、POP3和IMAP协议。

8.1.1.1.1.2. 请列举Nginx的一些特性?

Nginx服务器的特性包括：反向代理/L7负载均衡器 ；嵌入式Perl解释器 ；动态二进制升级；可用于重新编写URL，具有非常好的PCRE支持。

8.1.1.1.1.3. nginx和apache的区别?

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；最核心的区别在于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。

8.1.1.1.1.4. nginx是如何实现高并发的?

一个主进程，多个工作进程，每个工作进程可以处理多个请求，每进来一个request，会有一个worker进程去处理。但不是全程的处理，处理到可能发生阻塞的地方，比如向上游（后端）服务器转发request，并等待请求返回。那么，这个处理的worker继续处理其他请求，而一旦上游服务器返回了，就会触发这个事件，worker才会来接手，这个request才会接着往下走。由于web server的工作性质决定了每个request的大部份生命都是在网络传输中，实际上花费在server机器上的时间片不多。这是几个进程就解决高并发的秘密所在。即@skoo所说的webserver刚好属于网络io密集型应用，不算是计算密集型。

8.1.1.1.1.5. nginx为什么不使用多线程?

Nginx:采用单线程来异步非阻塞处理请求（管理员可以配置Nginx主进程的工作进程的数量），不会为每个请求分配cpu和内存资源，节省了大量资源，同时也减少了大量的CPU的上下文切换，所以才使得Nginx支持更高的并发。

8.1.1.1.1.6. 请解释Nginx服务器上的Master和Worker进程分别是什么?

主程序 Master process 启动后，通过一个 for 循环来 接收 和 处理外部信号 ；
主进程通过 fork() 函数产生 worker 子进程 ，每个子进程执行一个 for循环来实现Nginx服务器对事件的接收和处理 。
一般推荐 worker 进程数与CPU内核数一致，这样一来不存在大量的子进程生成和管理任务，避免了进程之间竞争CPU 资源和进程切换的开销。而且 Nginx 为了更好的利用 多核特性 ，提供了 CPU 亲缘性的绑定选项，我们可以将某一个进程绑定在某一个核上，这样就不会因为进程的切换带来 Cache 的失效。
对于每个请求，有且只有一个工作进程 对其处理。首先，每个 worker 进程都是从 master进程 fork 过来。在 master 进程里面，先建立好需要 listen 的 socket（listenfd） 之后，然后再 fork 出多个 worker 进程。
所有 worker 进程的 listenfd 会在新连接到来时变得可读 ，为保证只有一个进程处理该连接，所有 worker 进程在注册 listenfd 读事件前抢占 accept_mutex ，抢到互斥锁的那个进程注册 listenfd 读事件 ，在读事件里调用 accept 接受该连接。
当一个 worker 进程在 accept 这个连接之后，就开始读取请求、解析请求、处理请求，产生数据后，再返回给客户端 ，最后才断开连接。这样一个完整的请求就是这样的了。我们可以看到，一个请求，完全由 worker 进程来处理，而且只在一个 worker 进程中处理。
在 Nginx 服务器的运行过程中， 主进程和工作进程 需要进程交互。交互依赖于 Socket 实现的管道来实现。

8.1.1.1.1.7. 请问ngx_http_upstream_module的作用是什么?

ngx_http_upstream_module用于定义可通过fastcgi传递、proxy传递、uwsgi传递、memcached传递和scgi传递指令来引用的服务器组。

8.1.1.1.1.8. 请问什么是C10K问题?

C10K问题是指无法同时处理大量客户端(10,000)的网络套接字。

8.1.1.1.1.9. Nginx是否支持将请求压缩到上游?

您可以使用Nginx模块gunzip将请求压缩到上游。gunzip模块是一个过滤器，它可以对不支持“gzip”编码方法的客户机或服务器使用“内容编码:gzip”来解压缩响应。

8.1.1.1.1.10. 如何在Nginx服务器上添加模块?

在编译过程中，必须选择Nginx模块，因为Nginx不支持模块的运行时间选择。

8.1.1.1.1.11. nginx特性和功能

特性：
模块化设计、较好的扩展性
高可靠性：一个master启动一或多个worker,每个worker响应多个请求
低内存消耗：10000个keepalive连接在Nginx中仅消耗2.5MB内存（官方数据）
支持热部署：不停机更新配置文件、更新日志文件、更新服务器程序版本

功能：
静态web资源服务器，能够缓存打开的文件描述符
支持http/imap/pop3/smtp的反向代理；支持缓存、负载均衡
支持fastcgi(fpm)
模块化，非DSO机制，支持过滤器zip压缩，SSI以及图像大小调整
支持SSL

9. LVS+Keepalived企业级集群解决方案

9.1.1.1.1.1. lvs 原理

LVS通过工作于内核的ipvs模块来实现功能，其主要工作于netfilter 的INPUT链上。
而用户需要对ipvs进行操作配置则需要使用ipvsadm这个工具。
ipvsadm主要用于设置lvs模型、调度方式以及指定后端主机。

9.1.1.1.1.2. LVS 调度算法

静态方法：仅根据调度算法本身进行调度
rr: round robin，轮流，轮询，轮叫
wrr: weighted round robin，加权轮询
sh：source hashing，session绑定
dh: destination hashing，目标地址hash

动态方法：根据算法及各RS当前的负载状况进行调度
lc: least connection，最少连接
wlc: weighted lc，加权最少连接
sed: shortest expection delay，最少期望延迟
nq: never queue，永不排队
lblc: Locality-Based Least Connection，基于局部性的最少连接
lblcr：Replicated lblc，基于局部性的带复制功能的最少连接

LVS软件的工作模式及模式工作原理?

NET模式：安全性高，但是只通过网关接收和响应请求，负载低。
TUN模式：通过调度器接收用户请求，但是处理节点分布在因特网返回用户请求，这样代价昂贵。
DR模式：这种模式性能和成本相对都比较理想，半开式网络，节点和调度器都在局域网，而节点响应却通过路由返回给用户。

请描述LVS软件在企业架构中的组件位置及作用?

LVS一般在企业架构中最前面接收用户请求，它可以接收用户请求，然后根据算法把请求分配给比如nginx代理。充当整个架构的负载均衡的作用。

9.1.1.1.1.3. LVS软件的管理工具及模块名称?

ipvsadm
ip_vs

9.1.1.1.1.4. lvs和nginx的区别?

lvs的优点：
1.抗负载能力强，工作在第四层仅做分发之用，没有流量的产生，这个特点也决定了他在负责均衡软件里的性能最强，无流量，同时保证了均衡器IO的性能不会受到大流量的影响。

2.工作稳定，自身有完整的双机热备方案，如lvs+keepalived

3.应用范围比较广，可以对所有应用做负载均衡。

4.配置性比较低，这是一个缺点也是一个优点，因为没有可太多配置的东西，所以并不需要太多的接触，大大减少人为出错的几率。

lvs的缺点：
1.软件本身不支持正则处理，不能做动静分离。

2.如果网站应用比较庞大，lvs/DR+keepalived就比较复杂了，特别是后面有windows server应用的机器，实施及配置还有维护过程就比较麻烦，相对而言，nginx+keepalived就简单一点。

nginx的优点：
跨平台：可以linux系统运行，而且windows的移植版本。

配置简单：非常的简单，容易上手。

非阻塞并发连接；数据复制时，磁盘io的第一阶段是非阻塞，官方测试能支持5万并发连接，实际生产中能跑2-3万并发连接数。发送报文是，nginx一边介绍web服务器的返回数据，一边把数据发送给客户端浏览器。

自带简单检查：当有服务器宕机后，新的请求就不会发送到这台机器上了，而是发送到其他节点。

节省带宽：支持gzip压缩，开启浏览器缓存。

网络依赖性低，理论上能ping通就可以实现负载均衡，而且可以有效区分内网，外网流量。

内存消耗小，稳定性高：开启10个nginx消耗内存125m，可以很好的处理静态资源，内存消耗少，宕机率很低。

9.1.1.1.1.5. 裂脑问题产生的原因

1） 心跳链路故障，导致无法通信 2） 开启防火墙阻挡心跳信息传输 3） 心跳网卡地址配置不正确 4） 其他： 心跳方式不同，心跳广播冲突，软件bug等防止

9.1.1.1.1.6. 防止裂脑的方法

1）采用串行或以太网电缆连接，同时用两条心跳线路 2） 做好裂脑的监控报警，在问题发生时人为第一时间介入仲裁 3） 启用磁盘锁，即正在服务的一方只在发现心跳线全部断开时，才开启磁盘锁 4） fence设备（智能电源管理设备） 5） 增加仲裁盘 6） 加冗余线路

9.1.1.1.1.7. 请问Keepalived的功能

keepalived观其名可知，保持存活，在网络里面就是保持在线了，也就是所谓的高可用或热备，用来防止单点故障(单点故障是指一旦某一点出现故障就会导致整个系统架构的不可用)的发生，Keepalived通过请求一个vip来达到请求真是IP地址的功能，而VIP能够在一台机器发生故障时候，自动漂移到另外一台机器上，从来达到了高可用HA功能

请问Keepalived的原理

Keepalived工作在TCP/IP 参考模型的 三层、四层、五层，也就是分别为：网络层，传输层和应用层，根据TCP、IP参数模型隔层所能实现的功能，Keepalived运行机制如下：

在网络层：我们知道运行这4个重要的协议，互联网络IP协议，互联网络可控制报文协议ICMP、地址转换协议ARP、反向地址转换协议RARP，在网络层Keepalived在网络层采用最常见的工作方式是通过ICMP协议向服务器集群中的每一个节点发送一个ICMP数据包(有点类似与Ping的功能)，如果某个节点没有返回响应数据包，那么认为该节点发生了故障，Keepalived将报告这个节点失效，并从服务器集群中剔除故障节点。

在传输层：提供了两个主要的协议：传输控制协议TCP和用户数据协议UDP，传输控制协议TCP可以提供可靠的数据输出服务、IP地址和端口，代表TCP的一个连接端，要获得TCP服务，需要在发送机的一个端口和接收机的一个端口上建立连接，而Keepalived在传输层里利用了TCP协议的端口连接和扫描技术来判断集群节点的端口是否正常，比如对于常见的WEB服务器80端口。或者SSH服务22端口，Keepalived一旦在传输层探测到这些端口号没有数据响应和数据返回，就认为这些端口发生异常，然后强制将这些端口所对应的节点从服务器集群中剔除掉。

在应用层：可以运行FTP，TELNET，SMTP，DNS等各种不同类型的高层协议，Keepalived的运行方式也更加全面化和复杂化，用户可以通过自定义Keepalived工作方式，例如：可以通过编写程序或者脚本来运行Keepalived，而Keepalived将根据用户的设定参数检测各种程序或者服务是否允许正常，如果Keepalived的检测结果和用户设定的不一致时，Keepalived将把对应的服务器从服务器集群中剔除

9.1.1.1.1.8. keeplived 有哪些模块

主要有三个模块，分别是core、check和vrrp

9.1.1.1.1.9. 解释一下vrrp 协议

虚拟路由冗余协议，可以认为是实现路由器高可用的协议，即将N台提供相同功能的路由器组成一个路由器组，这个组里面有一个master和多个backup，master上面有一个对外提供服务的vip（该路由器所在局域网内其他机器的默认路由为该vip），master会发组播，当backup收不到vrrp包时就认为master宕掉了，这时就需要根据VRRP的优先级来选举一个backup当master。这样的话就可以保证路由器的高可用了。

9.1.1.1.1.10. keepalived作用

keepalived长和lvs或nginx搭配使用，进行负载均衡。

9.1.1.1.1.11. Keepalived的工作原理

在一个虚拟路由器中，只有作为MASTER的VRRP路由器会一直发送VRRP通告信息，
BACKUP不会抢占MASTER，除非它的优先级更高。当MASTER不可用时(BACKUP收不到通告信息)
多台BACKUP中优先级最高的这台会被抢占为MASTER。这种抢占是非常快速的(<1s)，以保证服务的连续性
由于安全性考虑，VRRP包使用了加密协议进行加密。BACKUP不会发送通告信息，只会接收通告信息

10. Zabbix企业级监控入门实战

10.1.1.1.1.1. 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

10.1.1.1.1.2. 哪些方面可以做监控，常见产品有?

```
1.硬件监控——路由器、交换机、防火墙
2.系统监控——cpu、内存、磁盘、网络、进程、tcp
3.服务监控——nginx、php、tomcat、redis、memcache、mysql
4.web监控——响应时间、加载时间、渲染时间
5.日志监控——ELK、（收集、存储、分析、展示）日志
6.安全监控——firewalld、WAF(nginx+lua)、安全宝、牛盾云、安全狗
```

10.1.1.1.1.3. 常见的监控项有哪些

```
单机进程cpu查看负载和使用率
单机内存查看
单机磁盘查看
单机查看网络
```

10.1.1.1.1.4. zabbix数据收集的模式

```
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。

主动模式被动模式：默认为zabbix-agent被动模式
主动模式与被动模式主要是站在zabbix-agent身份来说
1.被动模式（zabbix-server轮询检测zabbix-agent）
2.主动模式（zabbix-agent主动上报给zabbix-server）优
zabbix主动模式与被动模式选择
1.当（Queue）队列中有大量的延迟监控项
2.当监控主机超过300+ ,建议使用主动模式
```

10.1.1.1.1.5. zabbix 自定义发现是怎么做的

```
1、首先需要在模板当中创建一个自动发现的规则，这个地方只需要一个名称和一个键值。
2、过滤器中间要添加你需要的用到的值宏。
3、然后要创建一个监控项原型，也是一个名称和一个键值。
4、然后需要去写一个这样的键值的收集。
```

10.1.1.1.1.6. 邮件报警

10.1.1.1.1.7. 微信报警

```
1、首先，需要有一个微信企业号。（一个实名认证的[微信号]一个可以使用的[手机号]一个可以登录的[邮箱号]
2、下载并配置微信公众平台私有接口。
3、配置Zabbix告警，（增加示警媒介类型，添加用户报警媒介，添加报警动作）。
```

10.1.1.1.1.8. 自定义key

11. 高并发解决方案Haproxy及Nginx负载均衡集群实战

11.1.1.1.1.1. HAProxy在不影响现有连接的情况下，如何重新加载配置

```
使用haproxy的-sf(finished) 参数，可以方便重启了：
haproxy -f configfile -sf
haproxy -f /etc/haproxy/haproxy.cfg -p /var/run/haproxy.pid -sf $(cat
/var/run/haproxy.pid)
```

11.1.1.1.1.2. 你近期使用过的Nginx的版本?

```
生产环境使用Stable version：最新稳定版
注意各版本的区别：Nginx官网提供了三个类型的版本
1、Mainline version：Mainline 是 Nginx 目前主力在做的版本，可以说是开发版
2、Stable version：最新稳定版，生产环境上建议使用的版本
3、Legacy versions：遗留的老版本的稳定版
```

11.1.1.1.1.3. Nginx如何做限速限流?

```
# ip并发请求限制
limit_req_zone $binary_remote_addr zone=one:10m rate=50r/s;
limit_req_zone $server_name zone=perserver:10m rate=50r/s;
# ip 连接数限制
limit_conn_zone $binary_remote_addr zone=addr:10m;
limit_conn_zone $server_name zone=perserverconn:10m;
```

11.1.1.1.1.4. 如何对nginx中如下location pattern匹配的优先级排序

```
location  = / {}
location  / {}
location  /documents/ {}
location  ~* \.(gif|jpgljpeg)$ {}
location  ^~ /images/ {}

答：= 优先于 ^~  优先于于 ~|~*|!~|!~* 优先于 /
```

11.1.1.1.1.5. nginx代理负载均衡的调度算法有哪些?具体实现时的现象是什么?

```
轮询（默认）每个请求按时间顺序逐一分配到不同的后端服务，如果后端某台服务器死机，自动剔除故障系
统，使用户访问不受影响。
weight（轮询权值）weight的值越大分配到的访问概率越高，主要用于后端每台服务器性能不均衡的情况
下。或者仅仅为在主从的情况下设置不同的权值，达到合理有效的地利用主机资源。
ip_hash每个请求按访问IP的哈希结果分配，使来自同一个IP的访客固定访问一台后端服务器，并且可以有效
解决动态网页存在的session共享问题。
fair比 weight、ip_hash更加智能的负载均衡算法，fair算法可以根据页面大小和加载时间长短智能地进
行负载均衡，也就是根据后端服务器的响应时间 来分配请求，响应时间短的优先分配。Nginx本身不支持
fair，如果需要这种调度算法，则必须安装upstream_fair模块。
url_hash按访问的URL的哈希结果来分配请求，使每个URL定向到一台后端服务器，可以进一步提高后端缓存
服务器的效率。Nginx本身不支持url_hash，如果需要这种调度算法，则必须安装Nginx的hash软件包
```

11.1.1.1.1.6. 遇到前端nginx并发突然增高，写出你的排查思路

```
查看日志，看并发高的这段时间的访问记录以及出错记录等
进行访问限制
优化高并发配置
```

11.1.1.1.1.7. Nginx状态码499的意义是

499对应的是“client has closed connection”。这很有可能是因为服务器端处理的时间过长，客户端“不耐烦”了，测试nginx发现如果两次提交post过快就会出现499的情况，看来是nginx认为是不安全的连接，主动拒绝了客户端的连接.
配置参数 proxy_ignore_client_abort on;
表示代理服务端不要主要主动关闭客户端连接就可以了

11.1.1.1.1.8. LVS、HAProxy、Nginx三款

LVS的优点：
1、抗负载能力强、工作在第4层仅作分发之用，没有流量的产生，这个特点也决定了它在负载均衡软件里的性能最强的；无流量，同时保证了均衡器IO的性能不会受到大流量的影响；
2、工作稳定，自身有完整的双机热备方案，如LVS+Keepalived和LVS+Heartbeat；
3、应用范围比较广，可以对所有应用做负载均衡；
4、配置性比较低，这是一个缺点也是一个优点，因为没有可太多配置的东西，所以并不需要太多接触，大大减少了人为出错的几率；

LVS的缺点：
1、软件本身不支持正则处理，不能做动静分离，这就凸显了Nginx/HAProxy+Keepalived的优势。
2、如果网站应用比较庞大，LVS/DR+Keepalived就比较复杂了，特别是后面有Windows Server应用的机器，实施及配置还有维护过程就比较麻烦，相对而言，Nginx/HAProxy+Keepalived就简单多了。

Nginx的优点：
1、工作在OSI第7层，可以针对http应用做一些分流的策略。比如针对域名、目录结构。它的正则比HAProxy更为强大和灵活；
2、Nginx对网络的依赖非常小，理论上能ping通就就能进行负载功能，这个也是它的优势所在；
3、Nginx安装和配置比较简单，测试起来比较方便；
4、可以承担高的负载压力且稳定，一般能支撑超过几万次的并发量；
5、Nginx可以通过端口检测到服务器内部的故障，比如根据服务器处理网页返回的状态码、超时等等，并且会把返回错误的请求重新提交到另一个节点；
6、Nginx不仅仅是一款优秀的负载均衡器/反向代理软件，它同时也是功能强大的Web应用服务器。LNMP现在也是非常流行的web环境，大有和LAMP环境分庭抗礼之势，Nginx在处理静态页面、特别是抗高并发方面相对apache有优势；
7、Nginx现在作为Web反向加速缓存越来越成熟了，速度比传统的Squid服务器更快，有需求的朋友可以考虑用其作为反向代理加速器；

Nginx的缺点：
1、Nginx不支持url来检测。
2、Nginx仅能支持http和Email，这个它的弱势。
3、Nginx的Session的保持，Cookie的引导能力相对欠缺。

HAProxy的优点：
1、HAProxy是支持虚拟主机的，可以工作在4、7层(支持多网段)；
2、能够补充Nginx的一些缺点比如Session的保持，Cookie的引导等工作；
3、支持url检测后端的服务器；
4、它跟LVS一样，本身仅仅就只是一款负载均衡软件；单纯从效率上来讲HAProxy更会比Nginx有更出色的负载均衡速度，在并发处理上也是优于Nginx的；
5、HAProxy可以对Mysql读进行负载均衡，对后端的MySQL节点进行检测和负载均衡，不过在后端的MySQL slaves数量超过10台时性能不如LVS；
6、HAProxy的算法较多，达到8种；
LVS： 是基于四层的转发
HAproxy： 是基于四层和七层的转发，是专业的代理服务器
Nginx： 是WEB服务器，缓存服务器，又是反向代理服务器，可以做七层的转发

区别： LVS由于是基于四层的转发所以只能做端口的转发
而基于URL的、基于目录的这种转发LVS就做不了
工作选择：
HAproxy和Nginx由于可以做七层的转发，所以URL和目录的转发都可以做
在很大并发量的时候我们就要选择LVS，像中小型公司的话并发量没那么大
选择HAproxy或者Nginx足已，由于HAproxy由是专业的代理服务器
配置简单，所以中小型企业推荐使用HAproxy

11.1.1.1.1.9. 使用“反向代理服务器”的优点是什么?

反向代理服务器可以隐藏源服务器的存在和特征。它充当互联网云和web服务器之间的中间层。这对于安全方面来说是很好的，特别是当您使用web托管服务时。

11.1.1.1.1.10. 请列举Nginx服务器的最佳用途。

Nginx服务器的最佳用法是在网络上部署动态HTTP内容，使用SCGI、WSGI应用程序服务器、用于脚本的FastCGI处理程序。它还可以作为负载均衡器。

11.1.1.1.1.11. 为什么要做动、静分离?

在我们的软件开发中，有些请求是需要后台处理的（如：.jsp,.do等等），有些请求是不需要经过后台处理的（如：css、html、jpg、js等等），这些不需要经过后台处理的文件称为静态文件，否则动态文件。因此我们后台处理忽略静态文件，但是如果直接忽略静态文件的话，后台的请求次数就明显增多了。在我们对资源的响应速度有要求的时候，应该使用这种动静分离的策略去解决动、静分离将网站静态资源（HTML，JavaScript，CSS等）与后台应用分开部署，提高用户访问静态代码的速度，降低对后台应用访问。这里将静态资源放到nginx中，动态资源转发到tomcat服务器中,毕竟Tomcat的优势是处理动态请求。

12. Tomcat服务及Memcached Tomcat集群及JVM优化

12.1.1.1.1.1. 什么是Tomcat?

Tomcat简单的说就是一个运行JAVA的网络服务器，底层是Socket的一个程序，它也是JSP和Serlvet的一个容器。

12.1.1.1.1.2. 如何配置Tomcat虚拟目录?

1、在server.xml中的节点下添加如下代码。path表示的是访问时输入的web项目名，docBase表示的是站点目录的绝对路径。
2、进入到confCatalinalocalhost文件下，创建一个xml文件，该文件的名字就是站点的名字。

12.1.1.1.1.3. Tomcat体系结构是怎样的?

浏览器 -> tomcat server-> service ->connector -> engine(引擎) -> host(主机) -> web应用。

12.1.1.1.1.4. Tomcat的缺省端口是多少，怎么修改?

1)找到Tomcat目录下的conf文件夹;
2)进入conf文件夹里面找到server.xml文件;
3)打开server.xml文件;
4)在server.xml文件里面找到下列信息;
port="8080"改成你想要的端口

12.1.1.1.1.5. Tomcat 有几种部署方式?

1)直接把Web项目放在webapps下，Tomcat会自动将其部署

2)在server.xml文件上配置节点，设置相关的属性即可

3)通过Catalina来进行配置：进入到conf\Catalina\localhost文件下，创建一个xml文件，该文件的名字就是站点的名字。编写XML的方式来进行设置。

12.1.1.1.1.6. 什么是中间件？什么是jdk?

中间件是一种独立的系统软件或服务程序，分布式应用软件借助这种软件在不同的技术之间共享资源
中间件位于客户机/ 服务器的操作系统之上，管理计算机资源和网络通讯
是连接两个独立应用程序或独立系统的软件。相连接的系统，即使它们具有不同的接口

但通过中间件相互之间仍能交换信息。执行中间件的一个关键途径是信息传递
通过中间件，应用程序可以工作于多平台或OS环境。

jdk：jdk是Java的开发工具包
它是一种用于构建在 Java 平台上发布的应用程序、applet 和组件的开发环境

12.1.1.1.1.7. 讲述一下Tomcat8005、8009、8080三个端口的含义?

8005==》 关闭时使用
8009==》 为AJP端口，即容器使用，如Apache能通过AJP协议访问Tomcat的8009端口
8080==》 一般应用使用

12.1.1.1.1.8. Tomcat有几种部署方式?

1）直接把Web项目放在webapps下，Tomcat会自动将其部署
2）在server.xml文件上配置<Context>节点，设置相关的属性即可
3）通过Catalina来进行配置:进入到conf\Catalina\localhost文件下，创建一个xml文件，该文件的名字就是站点的名字。
编写XML的方式来进行设置。

12.1.1.1.1.9. tomcat容器是如何创建servlet类实例？用到了什么原理?

当容器启动时，会读取在webapps目录下所有的web应用中的web.xml文件，然后对xml文件进行解析，
并读取servlet注册信息。然后，将每个应用中注册的servlet类都进行加载，并通过反射的方式实例化。
（有时候也是在第一次请求时实例化）在servlet注册时加上如果为正数，则在一开始就实例化，
如果不写或为负数，则第一次请求实例化。

12.1.1.1.1.10. tomcat 如何优化?

1、优化连接配置.这里以tomcat7的参数配置为例，需要修改conf/server.xml文件，修改连接数，关闭客户端dns查询。

参数解释：
URIEncoding=”UTF-8” :使得tomcat可以解析含有中文名的文件的url，真方便，不像apache里还有搞个mod_encoding，还要手工编译
maxSpareThreads : 如果空闲状态的线程数多于设置的数目，则将这些线程中止，减少这个池中的线程总数。
minSpareThreads : 最小备用线程数，tomcat启动时的初始化的线程数。
enableLookups : 这个功效和Apache中的HostnameLookups一样，设为关闭。
connectionTimeout : connectionTimeout为网络连接超时时间毫秒数。
maxThreads : maxThreads Tomcat使用线程来处理接收的每个请求。这个值表示Tomcat可创建的最大的线程数，即最大并发数。
acceptCount : acceptCount是当线程数达到maxThreads后，后续请求会被放入一个等待队列，这个acceptCount是这个队列的大小，如果这个队列也满了，就直接refuse connection
maxProcessors与minProcessors : 在 Java中线程是程序运行时的路径，是在一个程序中与其它控制线程无关的、能够独立运行的代码段。它们共享相同的地址空间。多线程帮助程序员写出CPU最 大利用率的高效程序，使空闲时间保持最低，从而接受更多的请求。
通常Windows是1000个左右，Linux是2000个左右。
useURIValidationHack:

我们来看一下tomcat中的一段源码：

```
【security】

if (connector.getUseURIValidationHack()) {

String uri = validate(request.getRequestURI());

if (uri == null) {

res.setStatus(400);

res.setMessage(“Invalid URI”);

throw new IOException(“Invalid URI”);

} else {

req.requestURI().setString(uri);

// Redoing the URI decoding

req.decodedURI().duplicate(req.requestURI());

req.getURLDecoder().convert(req.decodedURI(), true);
```

可以看到如果把useURIValidationHack设成”false”，可以减少它对一些url的不必要的检查从而减省开销。

enableLookups=”false” : 为了消除DNS查询对性能的影响我们可以关闭DNS查询，方式是修改server.xml文件中的enableLookups参数值。

disableUploadTimeout : 类似于Apache中的keeyalive一样

给Tomcat配置gzip压缩(HTTP压缩)功能

compression=”on” compressionMinSize=”2048”

compressableMimeType=”text/html,text/xml,text/JavaScript,text/css,text/plain”

HTTP 压缩可以大大提高浏览网站的速度，它的原理是，在客户端请求网页后，从服务器端将网页文件压缩，再下载到客户端，由客户端的浏览器负责解压缩并浏览。相对于普通的浏览过程HTML,CSS,javascript , Text ，它可以节省40%左右的流量。更为重要的是，它可以对动态生成的，包括CGI、PHP ， JSP ， ASP ， Servlet,SHTML等输出的网页也能进行压缩，压缩效率惊人。

1)compression=”on” 打开压缩功能

2)compressionMinSize=”2048” 启用压缩的输出内容大小，这里面默认为2KB

3)noCompressionUserAgents=”gozilla, traviata” 对于以下的浏览器，不启用压缩

4)compressableMimeType=”text/html,text/xml” 压缩类型

最后不要忘了把8443端口的地方也加上同样的配置，因为如果我们走https协议的话，我们将会用到8443端口这个段的配置，对吧？

<!-enable tomcat ssl->

<Connector port=”8443” protocol=”HTTP/1.1”

URIEncoding=”UTF-8” minSpareThreads=”25” maxSpareThreads=”75”

enableLookups=”false” disableUploadTimeout=”true” connectionTimeout=”20000”

acceptCount=”300” maxThreads=”300” maxProcessors=”1000” minProcessors=”5”

useURIValidationHack=”false”

compression=”on” compressionMinSize=”2048”

compressableMimeType=”text/html,text/xml,text/javascript,text/css,text/plain”

SSLEnabled=”true”

scheme=”https” secure=”true”

clientAuth=”false” sslProtocol=”TLS”

keystoreFile=”d:/tomcat2/conf/shnlap93.jks” keystorePass=”aaaaaa”

/>

好了，所有的Tomcat优化的地方都加上了。

12.1.1.1.1.11. tomcat内存调优

内存方式的设置是在catalina.sh中，调整一下JAVA_OPTS变量即可，因为后面的启动参数会把JAVA_OPTS作为JVM的启动参数来处理。
具体设置如下：
```
JAVA_OPTS="$JAVA_OPTS -Xmx3550m -Xms3550m -Xss128k -XX:NewRatio=4 -
XX:SurvivorRatio=4"
```
其各项参数如下：
-Xmx3550m：设置JVM最大可用内存为3550M。
-Xms3550m：设置JVM促使内存为3550m。此值可以设置与-Xmx相同，以避免每次垃圾回收完成后JVM重新分配内存。
-Xmn2g：设置年轻代大小为2G。整个堆大小=年轻代大小 + 年老代大小 + 持久代大小。持久代一般固定大小为64m，所以增大年轻代后，将会减小年老代大小。此值对系统性能影响较大，Sun官方推荐配置为整个堆的3/8。
-Xss128k：设置每个线程的堆栈大小。JDK5.0以后每个线程堆栈大小为1M，以前每个线程堆栈大小为256K。更具应用的线程所需内存大小进行调整。在相同物理内存下，减小这个值能生成更多的线程。但是操作系统对一个进程内的线程数还是有限制的，不能无限生成，经验值在3000~5000左右。
-XX:NewRatio=4:设置年轻代（包括Eden和两个Survivor区）与年老代的比值（除去持久代）。设置为4，则年轻代与年老代所占比值为1：4，年轻代占整个堆栈的1/5
-XX:SurvivorRatio=4：设置年轻代中Eden区与Survivor区的大小比值。设置为4，则两个Survivor区与一个Eden区的比值为2:4，一个Survivor区占整个年轻代的1/6
-XX:MaxPermSize=16m:设置持久代大小为16m。
-XX:MaxTenuringThreshold=0：设置垃圾最大年龄。如果设置为0的话，则年轻代对象不经过Survivor区，直接进入年老代。对于年老代比较多的应用，可以提高效率。如果将此值设置为一个较大值，则年轻代对象会在Survivor区进行多次复制，这样可以增加对象再年轻代的存活时间，增加在年轻代即被回收的概论。

12.1.1.1.1.12. 垃圾回收策略调优

垃圾回收的设置也是在catalina.sh中，调整JAVA_OPTS变量。
具体设置如下：
```
JAVA_OPTS="$JAVA_OPTS -Xmx3550m -Xms3550m -Xss128k -XX:+UseParallelGC  -
XX:MaxGCPauseMillis=100"
```
具体的垃圾回收策略及相应策略的各项参数如下：
串行收集器（JDK1.5以前主要的回收方式）
-XX:+UseSerialGC:设置串行收集器
并行收集器（吞吐量优先）
示例：
```
java -Xmx3550m -Xms3550m -Xmn2g -Xss128k -XX:+UseParallelGC  -
XX:MaxGCPauseMillis=100
```

-XX:+UseParallelGC：选择垃圾收集器为并行收集器。此配置仅对年轻代有效。即上述配置下，年轻代使用并发收集，而年老代仍旧使用串行收集。
-XX:ParallelGCThreads=20：配置并行收集器的线程数，即：同时多少个线程一起进行垃圾回收。此值最好配置与处理器数目相等。
-XX:+UseParallelOldGC：配置年老代垃圾收集方式为并行收集。JDK6.0支持对年老代并行收集
-XX:MaxGCPauseMillis=100:设置每次年轻代垃圾回收的最长时间，如果无法满足此时间，JVM会自动调整年轻代大小，以满足此值。
-XX:+UseAdaptiveSizePolicy：设置此选项后，并行收集器会自动选择年轻代区大小和相应的Survivor区比例，以达到目标系统规定的最低相应时间或者收集频率等，此值建议使用并行收集器时，一直打开。
并发收集器（响应时间优先）
示例：java -Xmx3550m -Xms3550m -Xmn2g -Xss128k -XX:+UseConcMarkSweepGC
-XX:+UseConcMarkSweepGC：设置年老代为并发收集。测试中配置这个以后，-XX:NewRatio=4的配置失效了，原因不明。所以，此时年轻代大小最好用-Xmn设置。
-XX:+UseParNewGC: 设置年轻代为并行收集。可与CMS收集同时使用。JDK5.0以上，JVM会根据系统配置自行设置，所以无需再设置此值。
-XX:CMSFullGCsBeforeCompaction：由于并发收集器不对内存空间进行压缩、整理，所以运行一段时间以后会产生“碎片”，使得运行效率降低。此值设置运行多少次GC以后对内存空间进行压缩、整理。
-XX:+UseCMSCompactAtFullCollection：打开对年老代的压缩。可能会影响性能，但是可以消除碎片

12.1.1.1.1.13. 专业分析工具有哪些

IBM ISA，JProfiler、probe 等

12.1.1.1.1.14. 监视Tomcat的内存使用情况

使用JDK自带的jconsole可以比较明了的看到内存的使用情况，线程的状态，当前加载的类的总量等；
JDK自带的jvisualvm可以下载插件（如GC等），可以查看更丰富的信息。如果是分析本地的Tomcat的话，还可以进行内存抽样等，检查每个类的使用情况

13. Nosql数据库Redis企业级实战及VMwarevSphere

13.1.1.1.1.1. Redis 持久化机制

Redis是一个支持持久化的内存数据库，通过持久化机制把内存中的数据同步到硬盘文件来保证数据持久化。当Redis重启后通过把硬盘文件重新加载到内存，就能达到恢复数据的目的。
实现：单独创建fork()一个子进程，将当前父进程的数据库数据复制到子进程的内存中，然后由子进程写入到临时文件中，持久化的过程结束了，再用这个临时文件替换上次的快照文件，然后子进程退出，内存释放。

RDB是Redis默认的持久化方式。按照一定的时间周期策略把内存的数据以快照的形式保存到硬盘的二进制文件。即Snapshot快照存储，对应产生的数据文件为dump.rdb，通过配置文件中的save参数来定义快照的周期。（ 快照可以是其所表示的数据的一个副本，也可以是数据的一个复制品。）
AOF：Redis会将每一个收到的写命令都通过Write函数追加到文件最后，类似于MySQL的binlog。当Redis重启是会通过重新执行文件中保存的写命令来在内存中重建整个数据库的内容。
当两种方式同时开启时，数据恢复Redis会优先选择AOF恢复。

13.1.1.1.1.2. memcache的工作原理

memcache是以key-value形式存储的，key会通过一个hash表转换成hash的key，便于查找对比。客户端通过key的hash值确定数据的位置，然后向服务端发出请求，获取真实的数据

13.1.1.1.1.3. Memcache与Redis的区别

1)、存储方式 Memecache把数据全部存在内存之中，断电后会挂掉，数据不能超过内存大小。 Redis有部份存在硬盘上，redis可以持久化其数据
2)、数据支持类型 memcached所有的值均是简单的字符串，redis作为其替代者，支持更为丰富的数据类型，提供list，set，zset，hash等数据结构的存储
3)、使用底层模型不同 它们之间底层实现方式 以及与客户端之间通信的应用协议不一样。 Redis直接自己构建了VM 机制 ，因为一般的系统调用系统函数的话，会浪费一定的时间去移动和请求。
4). value 值大小不同：Redis 最大可以达到 512M；memcache 只有 1mb。
5）redis的速度比memcached快很多
6）Redis支持数据的备份，即master-slave模式的数据备份。

13.1.1.1.1.4. Redis的内存占用情况怎么样?

100万个键值对（键是0到999999值是字符串“hello world”）在我的32位的Mac笔记本上 用了100MB。同样的数据放到一个key里只需要16MB， 这是因为键值有一个很大的开销。 在Memcached上执行也是类似的结果，但是相对Redis的开销要小一点点，因为Redis会记录类型信息引用计数等等。

13.1.1.1.1.5. 单线程的redis为什么这么快

(一)纯内存操作
(二)单线程操作，避免了频繁的上下文切换
(三)采用了非阻塞I/O多路复用机制

13.1.1.1.1.6. Redis 为什么是单线程

因为Redis是基于内存的操作，CPU不是Redis的瓶颈，Redis的瓶颈最有可能是机器内存的大小或者网络带宽。既然单线程容易实现，而且CPU不会成为瓶颈，那就顺理成章地采用单线程的方案了（毕竟采用多线程会有很多麻烦！）Redis利用队列技术将并发访问变为串行访问
1）绝大部分请求是纯粹的内存操作（非常快速）2）采用单线程,避免了不必要的上下文切换和竞争条件
3）非阻塞IO优点：
1.速度快，因为数据存在内存中，类似于HashMap，HashMap的优势就是查找和操作的时间复杂度都是O(1)
2. 支持丰富数据类型，支持string，list，set，sorted set，hash
3.支持事务，操作都是原子性，所谓的原子性就是对数据的更改要么全部执行，要么全部不执行
4. 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如何解决redis的并发竞争key问题

同时有多个子系统去set一个key。这个时候要注意什么呢？ 不推荐使用redis的事务机制。因为我们的生产环境，基本都是redis集群环境，做了数据分片操作。你一个事务中有涉及到多个key操作的时候，这多个key不一定都存储在同一个redis-server上。因此，redis的事务机制，十分鸡肋。
(1)如果对这个key操作，不要求顺序： 准备一个分布式锁，大家去抢锁，抢到锁就做set操作即可
(2)如果对这个key操作，要求顺序： 分布式锁+时间戳。 假设这会系统B先抢到锁，将key1设置为{valueB 3:05}。接下来系统A抢到锁，发现自己的valueA的时间戳早于缓存中的时间戳，那就不做set操作了。以此类推。
(3) 利用队列，将set方法变成串行访问也可以redis遇到高并发，如果保证读写key的一致性
对redis的操作都是具有原子性的,是线程安全的操作,你不用考虑并发问题,redis内部已经帮你处理好并发的问题了。

13.1.1.1.1.7. redis的数据类型，以及每种数据类型的使用场景

(一)String
这个其实没啥好说的，最常规的set/get操作，value可以是String也可以是数字。一般做一些复杂的计数功能的缓存。
(二)hash
这里value存放的是结构化的对象，比较方便的就是操作其中的某个字段。博主在做单点登录的时候，就是用这种数据结构存储用户信息，以cookieId作为key，设置30分钟为缓存过期时间，能很好的模拟出类似session的效果。
(三)list
使用List的数据结构，可以做简单的消息队列的功能。另外还有一个就是，可以利用lrange命令，做基于redis的分页功能，性能极佳，用户体验好。本人还用一个场景，很合适—取行情信息。就也是个生产者和消费者的场景。LIST可以很好的完成排队，先进先出的原则。
(四)set
因为set堆放的是一堆不重复值的集合。所以可以做全局去重的功能。为什么不用JVM自带的Set进行去重？因为我们的系统一般都是集群部署，使用JVM自带的Set，比较麻烦，难道为了一个做一个全局去重，再起一个公共服务，太麻烦了。
另外，就是利用交集、并集、差集等操作，可以计算共同喜好，全部的喜好，自己独有的喜好等功能。
(五)sorted set
sorted set多了一个权重参数score,集合中的元素能够按score进行排列。可以做排行榜应用，取TOP N操作。

13.1.1.1.1.8. redis的过期策略以及内存淘汰机制

redis采用的是定期删除+惰性删除策略。
为什么不用定时删除策略?
定时删除,用一个定时器来负责监视key,过期则自动删除。虽然内存及时释放，但是十分消耗CPU资源。在大并发请求下，CPU要将时间应用在处理请求，而不是删除key,因此没有采用这一策略.
定期删除+惰性删除是如何工作的呢?
定期删除，redis默认每个100ms检查，是否有过期的key,有过期key则删除。需要说明的是，redis不是每个100ms将所有的key检查一次，而是随机抽取进行检查(如果每隔100ms,全部key进行检查，redis岂不是卡死)。因此，如果只采用定期删除策略，会导致很多key到时间没有删除。
于是，惰性删除派上用场。也就是说在你获取某个key的时候，redis会检查一下，这个key如果设置了过期时间那么是否过期了？如果过期了此时就会删除。
采用定期删除+惰性删除就没其他问题了么?
不是的，如果定期删除没删除key。然后你也没即时去请求key，也就是说惰性删除也没生效。这样，redis的内存会越来越高。那么就应该采用内存淘汰机制。

在redis.conf中有一行配置
maxmemory-policy volatile-lru

13.1.1.1.1.9. Reids6种淘汰策略

noeviction: 不删除策略，达到最大内存限制时，如果需要更多内存，直接返回错误信息。大多数写命令都会导致占用更多的内存(有极少数会例外。
allkeys-lru:所有key通用; 优先删除最近最少使用(less recently used ,LRU) 的 key。
volatile-lru:只限于设置了 expire 的部分; 优先删除最近最少使用(less recently used ,LRU) 的 key。
allkeys-random:所有key通用; 随机删除一部分 key。
volatile-random: 只限于设置了 ****expire**** 的部分; 随机删除一部分 key。
volatile-ttl: 只限于设置了 expire 的部分; 优先删除剩余时间(time to live,TTL) 短的key。

13.1.1.1.1.10. Redis内存模型

used_memory：Redis分配器分配的内存总量（单位是字节），包括使用的虚拟内存（即swap）；used_memory_human只是显示更友好。
used_memory_rss：Redis进程占据操作系统的内存（单位是字节），与top及ps命令看到的值是一致的；除了分配器分配的内存之外，used_memory_rss还包括进程运行本身需要的内存、内存碎片等，但是不包括虚拟内存。
mem_fragmentation_ratio：内存碎片比率，该值是used_memory_rss used_memory的比值。
mem_allocator：Redis使用的内存分配器，在编译时指定；可以是 libc 、jemalloc或者tcmalloc，默认是jemalloc；截图中使用的便是默认的jemalloc。

13.1.1.1.1.11. Redis内存划分

数据：作为数据库，数据是最主要的部分；这部分占用的内存会统计在used_memory中。

进程本身运行需要的内存：Redis主进程本身运行肯定需要占用内存，如代码、常量池等等；这部分内存大约几兆，在大多数生产环境中与Redis数据占用的内存相比可以忽略。这部分内存不是由jemalloc分配，因此不会统计在used_memory中。

缓冲内存：缓冲内存包括客户端缓冲区、复制积压缓冲区、AOF缓冲区等；其中，客户端缓冲存储客户端连接的输入输出缓冲；复制积压缓冲用于部分复制功能；AOF缓冲区用于在进行AOF重写时，保存最近的写入命令。在了解相应功能之前，不需要知道这些缓冲的细节；这部分内存由jemalloc分配，因此会统计在used_memory中。

内存碎片：内存碎片是Redis在分配、回收物理内存过程中产生的。例如，如果对数据的更改频繁，而且数据之间的大小相差很大，可能导致redis释放的空间在物理内存中并没有释放，但redis又无法有效利用，这就形成了内存碎片。内存碎片不会统计在used_memory中。

13.1.1.1.1.12. RDB和AOF的优缺点

RDB持久化
优点：RDB文件紧凑，体积小，网络传输快，适合全量复制；恢复速度比AOF快很多。当然，与AOF相比，RDB最重要的优点之一是对性能的影响相对较小。
缺点：RDB文件的致命缺点在于其数据快照的持久化方式决定了必然做不到实时持久化，而在数据越来越重要的今天，数据的大量丢失很多时候是无法接受的，因此AOF持久化成为主流。此外，RDB文件需要满足特定格式，兼容性差（如老版本的Redis不兼容新版本的RDB文件）。

AOF持久化
与RDB持久化相对应，AOF的优点在于支持秒级持久化、兼容性好，缺点是文件大、恢复速度慢、对性能影响大。

13.1.1.1.1.13. 持久化策略选择

（1）如果Redis中的数据完全丢弃也没有关系（如Redis完全用作DB层数据的cache），那么无论是单机，还是主从架构，都可以不进行任何持久化。

（2）在单机环境下（对于个人开发者，这种情况可能比较常见），如果可以接受十几分钟或更多的数据丢失，选择RDB对Redis的性能更加有利；如果只能接受秒级别的数据丢失，应该选择AOF。

（3）但在多数情况下，我们都会配置主从环境，slave的存在既可以实现数据的热备，也可以进行读写分离分担Redis读请求，以及在master宕掉后继续提供服务。

13.1.1.1.1.14. 为什么需要持久化?

由于Redis是一种内存型数据库，即服务器在运行时，系统为其分配了一部分内存存储数据，一旦服务器挂了，或者突然宕机了，那么数据库里面的数据将会丢失，为了使服务器即使突然关机也能保存数据，必须通过持久化的方式将数据从内存保存到磁盘中。

13.1.1.1.1.15. Redis 集群方案应该怎么做？都有哪些方案?

1.twemproxy，大概概念是，它类似于一个代理方式， 使用时在本需要连接 redis 的地方改为连接 twemproxy， 它会以一个代理的身份接收请求并使用一致性 hash 算法，将请求转接到具体 redis，将结果再返回 twemproxy。
缺点： twemproxy 自身单端口实例的压力，使用一致性 hash 后，对 redis 节点数量改变时候的计算值的改变，数据无法自动移动到新的节点。

2.codis，目前用的最多的集群方案，基本和 twemproxy 一致的效果，但它支持在 节点数量改变情况下，旧节点数据可恢复到新 hash 节点

3.redis cluster3.0 自带的集群，特点在于他的分布式算法不是一致性 hash，而是 hash 槽的概念，以及自身支持节点设置从节点。具体看官方文档介绍。

13.1.1.1.1.16. redis哨兵模式的特性和特点以及缺点。

这个哨兵模式的优点呢有很多，比如说这个支持的数据类型多，可以用AOF或RDB持久化存储，他性能还是特别好的，因为他用的是这个全内存操作嘛，至于这个缺点呢它主要是依赖于硬件设备，比如说他只能使用单线程，这个性能呢是受限于CPU性能的还有就是这个数据的存储量是跟机器内存大小相关的。

13.1.1.1.1.17. 什么是缓存穿透，什么是缓存雪崩

缓存穿透:
当一个微应用或者分布式环境有上百万的流量请求时,如果这些请求的数据 在redis缓存这一层不存在,那么就会穿过redis的缓存直达到后台,也就是mysql服务器,导致整个微服务应用挂掉
缓存雪崩:
就是redis缓存直接挂掉了,请求穿过缓存直接到达数据库,数据的并发访问量为几千,几万的访问量,导致数据直接挂掉,最终导致整个系统挂掉

13.1.1.1.1.18. MySQL与MongoDB之间最基本的差别是什么?

MySQL和MongoDB两者都是免费开源的数据库。MySQL和MongoDB有许多基本差别包括数据的表示(data representation)，查询，关系，事务，schema的设计和定义，标准化(normalization)，速度和性能。通过比较MySQL和MongoDB，实际上我们是在比较关系型和非关系型数据库，即数据存储结构不同。

13.1.1.1.1.19. MongoDB更新操作立刻fsync到磁盘?

不会，磁盘写操作默认是延迟执行的。写操作可能在两三秒(默认在60秒内)后到达磁盘。例如，如果一秒内数据库收到一千个对一个对象递增的操作，仅刷新磁盘一次。

13.1.1.1.1.20. 为什么MongoDB的数据文件巨大

MongoDB会积极的预分配预留空间来防止文件系统碎片。

13.1.1.1.1.21. MongoDB支持存储过程吗？如果支持的话，怎么用?

MongoDB支持存储过程，它是javascript写的，保存在db.system.js表中。

13.1.1.1.1.22. 如何理解MongoDB中的GridFS机制，MongoDB为何使用GridFS来存储文件?

GridFS是一种将大型文件存储在MongoDB中的文件规范。使用GridFS可以将大文件分隔成多个小文档存放，这样我们能够有效的保存大文档，而且解决了BSON对象有限制的问题。

14. 虚拟化技术与KVM企业级应用

14.1.1.1.1.1. 为什么要用虚拟化

充分利用物理资源，提供冗余性、向云计算演进的必要基础。

14.1.1.1.1.2. CPU虚拟化

CPU的虚拟化本质上是时间分片。这里我们用的是KVM来虚拟的。流程如下：
QEMU调用ioctl进入内核，内核读取VMCS，（这里先解释下VMCS，VMCS是异常处理的关键结构，里面存储了客户机，宿主机，陷入退出的各种状态。）把客户机的状态装在到CPU的CS:EIP，然后执行kvm_exit_handler. 如果是IO引起的，则进入用户模式，执行QEMU.

14.1.1.1.1.3. 内存虚拟化

一般的内存虚拟化都经过了两次的页表转换。 客户机线性地址到 客户机物理地址， 然后再到宿主机物理地址。 这样经过了两次转换，效率很难提高。于是就出现了影子页表。
这里就不得不说影子页表了。 这里有一张hash表，维护者物理机和客户机的内存地址映射表。 那么如何被建立的呢？

当发生页故障的时候，VMM搜索客户页表，如果不存在客户线性地址VGA 到 客户机物理地址 GPA的转换，那么则返回给虚拟机异常，由虚拟机处理页故障。
如果存在映射关系，则先去寻找QEMU地址空间的主机线性地址 HVA， 如果不存在 HVA 映射到 HPA， 那么 kernel 去分配物理地址。 然后VMM可以使用该HPA构建影子页表。 至此，影子页表建立完成。

14.1.1.1.1.4. 磁盘使用raw/qcow2有什么区别

raw不支持快照但性能好，常用的是qcom2支持快照，性能相比差一点。

14.1.1.1.1.5. kvm和Xen有何区别?

Xen是一个外部的hypervisor程序(虚拟机管理程序);它能够控制虚拟机和给多个客户机分配资源. 另一方面, kvm是linux的一部分, 可使用通常的linux调度器和内存管理. 这意味着kvm更小更易使用.
另一方面, Xen同时支持全虚拟化和半虚拟化(修改过的客户机能有更好的性能). kvm当前不支持半虚拟化.

14.1.1.1.1.6. 如果对一个VM进程使用kill -9将会发生什么?

从客户机的角度来看，就如你猛地把电源线从主机上拔出一样. 从主机的角度来看，进程被杀掉，进程占用的所有资源被施放.

14.1.1.1.1.7. kvm各模式说明

有三种模式：用户模式，内核模式，客户模式。 QEMU运行在用户模式，用户模式就是一般意义上运行的进程的模式，QEMU调用一系列的ioctl就可以进入内核模式，这里内核模式再调用kvm_create等函数就可以进入客户模式，客户模式也就是虚拟机运行的模式。

但是如果虚拟机出现了异常或者常见的缺页中断或者IO操作的时候呢，他是没有权限对全局资源进行操作的，这样就会被VMM，这里也就是KVM进行捕获，然后捕获之后呢再跳转到用户模式调用QEMU进行IO操作。所以说内核模式其实是个中间变量，专门用来进行错误发现的时候的转换的。

14.1.1.1.1.8. kvm框架包含哪些

KVM 分为了两个模块，一个是KVM Driver，另一个是QEMU。前者负责CPU和内存的虚拟化，后者负责IO的虚拟化。用户空间的QEMU调用一系列的ioctl() 函数进入内核空间，然后内核控件再调用一系列函数进入Guest OS客户空间运行虚拟机。

14.1.1.1.1.9. kvm的三个组件及作用

libvirt（用来管理虚拟机）、virt（安装和克隆虚拟机）、qemu（管理虚拟机磁盘的）

14.1.1.1.1.10. 简述kvm的几种模式

有三种模式：用户模式，内核模式，客户模式。 QEMU运行在用户模式，用户模式就是一般意义上运行的进程的模式，QEMU调用一系列的ioctl就可以进入内核模式，这里内核模式再调用kvm_create等函数就可以进入客户模式，客户模式也就是虚拟机运行的模式。

但是如果虚拟机出现了异常或者常见的缺页中断或者IO操作的时候呢，他是没有权限对全局资源进行操作的，这样就会被VMM，这里也就是KVM进行捕获，然后捕获之后呢再跳转到用户模式调用QEMU进行IO操作。所以说内核模式其实是个中间变量，专门用来进行错误发现的时候的转换的。

14.1.1.1.1.11. kvm如何做CPU虚拟化

CPU的虚拟化本质上是时间分片。这里我们用的是KVM来虚拟的。流程如下：
QEMU调用ioctl进入内核，内核读取VMCS，（这里先解释下VMCS，VMCS是异常处理的关键结构，里面存储了客户机，宿主机，陷入退出的各种状态。）把客户机的状态装在到CPU的CS:EIP，然后执行kvm_exit_handler. 如果是IO引起的，则进入用户模式，执行QEMU.

14.1.1.1.1.12. 如何对KVM的I/O优化

调优kvm的I/O调度算法，centos7默认的是deadline，使用命令将参数改为noop并查询，将以上的命令和查询结果以文本形式提交到答题框。

```
[root@localhost ~]# cat /sys/block/vda/queue/scheduler
[root@localhost ~]# echo noop > /sys/block/vda/queue/scheduler
[root@localhost ~]# cat /sys/block/vda/queue/scheduler
```

14.1.1.1.1.13. 如何对KVM内存调优

Kvm大页：所谓的大页指的是内存的大页面，内存采用的是分页机制，内存默认的页面大小都是4KB，假如系统里的一个应用程序需要2MB的内容，如果操作系统还是以4KB小页为单位，那么内存里就要有512个页面（512*4KB=2M），所以在TLB里就需要512个表项以及512个页表项，因此操作系统就要经历512次的TLB miss和512次的缺页中断才能将2MB的应用程序空间全部映射到物理内存里。必然操作数量会大大增加，从而间接的影响性能。

使用cat命令，只查看当前系统有多少大页，然后设置大页数量为20并查看，接着使用命令使配置永久生效，最后将大页挂载到/dev/hugepages/上。将上述所有命令和返回结果以文本形式提交到答题框。。

```
[root@localhost ~]# cat /proc/meminfo | grep HugePages
[root@localhost ~]# echo 20 >/proc/sys/vm/nr_hugepages
[root@localhost ~]# cat /proc/meminfo |grep Hugepages
[root@localhost ~]# sysctl -w vm.nr_HugePages=20
[root@localhost ~]# mount -t hugetlbfs huagetlbfs /dev/huagepages/
```

15. OpenVPN与JumpServer实战

15.1.1.1.1.1. 简述VPN，常见有哪几种?

VPN是指在公共的网络上建立专用网络的技术，但是两个节点间并没有物理上的专用的端到端链路，而是通过广域网或者运营商提供的网络平台之上的逻辑网络，用户数据在逻辑链路中传输，它可以有效的节省一般需要达到DDN专线所能达到的同样的目的，而且VPN采用身份验证和加密技术，充分保证了安全性。常见的VPN有：ipsec vpn、PPTP vpn、L2TP vpn、SSL vpn

JumpServer是什么

是一款使用 Python, Django 开发的开源跳板机系统，为互联网企业提供了认证，授权，审计，自动化运维等功能。

16. 阿里云或腾讯云云服务企业级实战

ECS

RDS

SLB

VPC

OSS

浮动IP

17. 分布式存储系统Ceph

17.1.1.1.1.1. fastDFS的角色

```
tracker server:跟踪服务器,主要做调度工作,起到均衡的作用;负责管理所有的 storage server
和 group,每个 storage 在启动后会连接 Tracker,告知自己所属 group 等信息,并保持周期性心跳,

storage server:存储服务器,主要提供容量和备份服务;以 group 为单位,每个 group 内可以包
含多台 storage server,数据互为备份,存储容量空间以 group 内容量最小的 storage 为准;建
议 group 内的 storage server 配置相同;以 group 为单位组织存储能够方便的进行应用隔离、负
载均衡和副本数定制;缺点是 group 的容量受单机存储容量的限制,同时 group 内机器坏掉,数据
恢复只能依赖 group 内其他机器重新同步。

client：客户端
```

17.1.1.1.1.2. fastDFS多个 group 之间的存储方式

```
Round robin,所有 group 轮询使用
Specified group,指定某个确定的 group
Load balance,剩余存储空间较多的 group 优先
```

17.1.1.1.1.3. fastDFS的同步机制

```
由于 storage server 上配置了所有的 tracker server,storage server 和 tracker
server 之间的通信是由 storage server 主动发起的,storage server 为每个 tracker
server 启动一个线程进行通信;在通信过程中,若发现该 tracker server 返回的本组
storage server 列表比本机记录少,就会将该 tracker server 上没有的 storage server
同步给该 tracker,这样的机制使得 tracker 之间是对等关系,数据保持一致。
```

17.1.1.1.1.4. fastDFS新增 storage 服务器数据同步

```
若新增 storage server 或者其状态发生变化,tracker server 都会将 storage server
列表同步给该组内所有 storage server;以新增 storage server 为例,因为新加入的
storage server 会主动连接 tracker server,tracker server 发现有新的 storage server
加入,就会将该组内所有的 storage server 返回给新加入的 storage server,并重新将
该组的 storage server 列表返回给该组内的其他 storage server。
```

17.1.1.1.1.5. ceph有哪几种接口

```
Object：有原生的API，而且也兼容Swift和S3的API。
Block：支持精简配置、快照、克隆。
File：Posix接口，支持快照。
```

17.1.1.1.1.6. ceph特性

```
集群可靠性
    尽可能的保障数据不会丢失。
    数据写入过程中不会因为意外情况出现而造成数据丢失。
    降低不可控物理因素造成的数据丢失。例如死机、断电等不可控物理因素。
集群可扩展性
```

系统规模可扩展。
存储容量可扩展。
随着系统节点数增加的聚合数据访问带宽的线性扩展。
数据安全性
保障不可控物理因素（死机、断电等）自然因素的生产、数据不会丢失，并且支持数据自动回复，自动平衡等。
保证系统规模扩大以后，运维难度保持在一个相对较低的水平。
接口统一性
同时支持三种存储：块存储、对象存储和文件存储。
支持市面上所有流行的存储类型。
去除所有的中心节点，防止单点故障

17.1.1.1.1.7. ceph系统组件

RADOS(Reliable Autonomic Object Store，可靠、自动、分布式对象存储)
OSD(对象存储设备)
MON(Ceph monitor)
RBD(RADOS块设备、Ceph块设备)
RGW(RADOS网关、Ceph对象网关)
MDS(Ceph元数据服务器)
CephFS(Ceph文件系统)

17.1.1.1.1.8. CRUSH算法

CRUSH算法通过每个设备的权重来计算数据对象的分布。对象分布是由cluster map和data distribution policy决定的。cluster map描述了可用存储资源和层级结构(比如有多少个机架，每个机架上有多少个服务器，每个服务器上有多少个磁盘)。data distribution policy由 placement rules组成。rule决定了每个数据对象有多少个副本，这些副本存储的限制条件(比如3个副本放在不同的机架中)。

17.1.1.1.1.9. CRUSH算法优点

任何组件都可以独立计算出每个object所在的位置(去中心化)。
只需要很少的元数据(cluster map)，只要当删除添加设备时，这些元数据才需要改变。

18. Zabbix分布式拓展和插件定制及Prometheus监控

18.1.1.1.1.1. 监控能做什么

硬件、软件、意外故障、关键事件、监控系统、趋势数据、报警灯

18.1.1.1.1.2. 自动发现

通过制定需要扫描的IP范围及时间间隔，使用规定的检查方法（端口/agent/system.uname）进行自动发现，
需要注意，自动发现含有VIP的主机，可能会产生重复添加，按需处理。

18.1.1.1.1.3. Prometheus 的局限

- Prometheus 是基于 Metric 的监控，不适用于日志（Logs）、事件(Event)、调用链(Tracing)。
- Prometheus 默认是 Pull 模型，合理规划你的网络，尽量不要转发。
- 对于集群化和水平扩展，官方和社区都没有银弹，需要合理选择 Federate、Cortex、Thanos等方案。
- 监控系统一般情况下可用性大于一致性，容忍部分副本数据丢失，保证查询请求成功。这个后面说 Thanos 去重的时候会提到。
- Prometheus 不一定保证数据准确，这里的不准确一是指 rate、histogram_quantile 等函数会做统计和推断，产生一些反直觉的结果，这个后面会详细展开。二来查询范围过长要做降采样，势必会造成数据精度丢失，不过这是时序数据的特点，也是不同于日志系统的地方。

19. 消息队列RabbitMQ与微服务Dubbo、Maven、 Nexus

19.1.1.1.1.1. 为什么使用消息队列

异步：批量数据异步处理（批量上传文件）
削峰：高负载任务负载均衡（电商秒杀抢购）
解耦：串行任务并行化（退货流程解耦）
广播：基于Pub/Sub实现一对多通信
蓄流压测（线上有些链路不好压测，可以通过堆积一定量消息再放开来压测）

19.1.1.1.1.2. RabbitMQ 集群部署

普通模式：创建好RabbitMQ 集群之后的默认模式。
镜像模式：把需要的队列做成镜像队列。

19.1.1.1.1.3. 如何查看RabbitMQ的集群状态

```
rabbitmqctl  cluster_status
```

19.1.1.1.1.4. ZooKeeper 使用场景

ZooKeeper 是一个分布式服务框架，它主要是用来解决分布式应用中经常遇到的一些数据管理问题，如：命名服务、状态同步、配置中心、集群管理等。

19.1.1.1.1.5. kafka 优势

通过O(1)的磁盘数据结构提供消息的持久化，这种结构对于即使数以 TB的消息存储也能够保持长时间的稳定性能。
高吞吐量：即使是非常普通的硬件Kafka 也可以支持每秒数百万的消息。
支持通过Kafka 服务器分区消息。
支持Hadoop 并行数据加载。

19.1.1.1.1.6. kafka存在几种角色

```
Broker：Kafka 集群包含一个或多个服务器，这种服务器被称为broker。
```

20. ELK日志收集与搜索分析系统实战

20.1.1.1.1.1. ELK是什么?

```
ELK 其实并不是一款软件，而是一整套解决方案，是三个软件产品的首字母缩写
Elasticsearch：负责日志检索和储存
Logstash：负责日志的收集和分析、处理
Kibana：负责日志的可视化
这三款软件都是开源软件，通常是配合使用，而且又先后归于 Elastic.co 公司名下，故被简称为 ELK
```

20.1.1.1.1.2. elasticsearch主要特点

```
1.实时分析
2.分布式实时文件存储，并将每一个字段都编入索引
3.文档导向，所有的对象全部是文档
4.高可用性，易扩展，支持集群（Cluster）、分片和复制（Shards 和 Replicas）
接口友好，支持 JSON
```

20.1.1.1.1.3. ES与关系数据库对比

```
ES 与关系型数据库的对比
在 ES 中，文档归属于一种 类型 (type) ，而这些类型存在于索引 (index) 中，类比传统关系型数据库
DB -> Databases -> Tables -> Rows -> Columns
关系型 数据库 表 行 列

ES -> Indices -> Types -> Documents -> Fields
ES 索引 类型 文档 域（字段）
```

20.1.1.1.1.4. ES 常用插件

```
head 插件：
它展现ES集群的拓扑结构，并且可以通过它来进行索引（Index）和节点（Node）级别的操作
它提供一组针对集群的查询API，并将结果以json和表格形式返回
它提供一些快捷菜单，用以展现集群的各种状态

kopf 插件
是一个ElasticSearch的管理工具
它提供了对ES集群操作的API

bigdesk 插件
是elasticsearch的一个集群监控工具
可以通过它来查看es集群的各种状态，如：cpu、内存使用情况，索引数据、搜索情况，http连接数等
```

20.1.1.1.1.5. kibana是什么及其特点

数据可视化平台工具

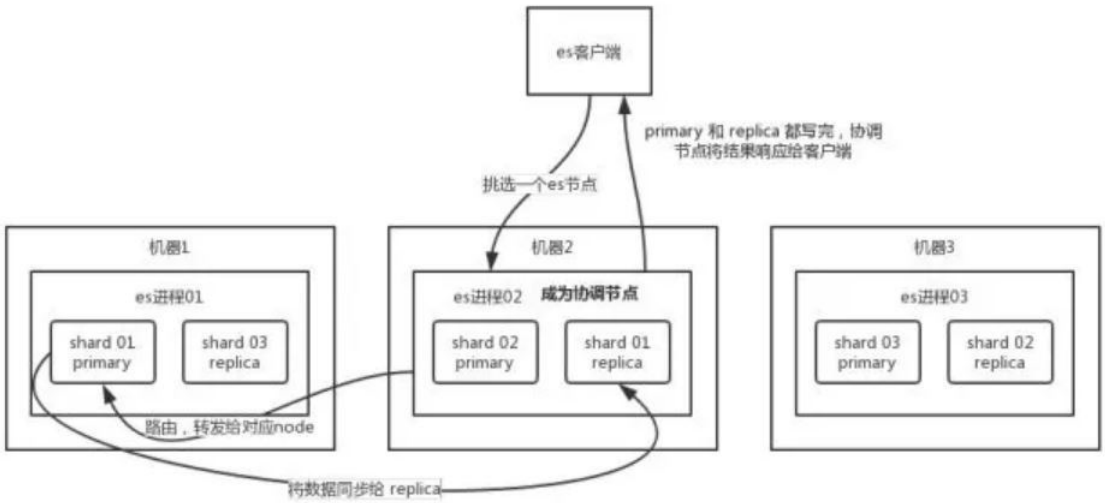
灵活的分析和可视化平台
实时总结和流数据的图表
为不同的用户显示直观的界面
即时分享和嵌入的仪表板

20.1.1.1.1.6. logstash 特点

所有类型的数据集中处理
不同模式和格式数据的正常化
自定义日志格式的迅速扩展
为自定义数据源轻松添加插件

20.1.1.1.1.7. ES写数据过程

客户端选择一个 node 发送请求过去，这个 node 就是 coordinating node（协调节点）。
coordinating node 对 document 进行路由，将请求转发给对应的 node（有 primary shard）。
[路由的算法是？]
实际的 node 上的 primary shard 处理请求，然后将数据同步到 replica node。
coordinating node 如果发现 primary node 和所有 replica node 都搞定之后，就返回响应结果给客户端。



20.1.1.1.1.8. es 读数据过程

可以通过 doc id 来查询，会根据 doc id 进行 hash，判断出来当时把 doc id 分配到了哪个 shard 上面去，从那个 shard 去查询。
客户端发送请求到任意一个 node，成为 coordinate node。
coordinate node 对 doc id 进行哈希路由，将请求转发到对应的 node，此时会使用 round-robin随机轮询算法，在 primary shard 以及其所有 replica 中随机选择一个，让读请求负载均衡。
接收请求的 node 返回 document 给 coordinate node。
coordinate node 返回 document 给客户端。

写请求是写入 primary shard，然后同步给所有的 replica shard；读请求可以从 primary shard 或 replica shard 读取，采用的是随机轮询算法。

20.1.1.1.1.9. es 搜索数据过程

20.1.1.1.1.10. Elasticsearch是如何实现Master选举的

Elasticsearch的选主是ZenDiscovery模块负责的，主要包含Ping（节点之间通过这个RPC来发现彼此）和Unicast（单播模块包含一个主机列表以控制哪些节点需要ping通）这两部分；

对所有可以成为master的节点（node.master: true）根据nodeId字典排序，每次选举每个节点都把自己所知道节点排一次序，然后选出第一个（第0位）节点，暂且认为它是master节点。

如果对某个节点的投票数达到一定的值（可以成为master节点数n/2+1）并且该节点自己也选举自己，那这个节点就是master。否则重新选举一直到满足上述条件。

master节点的职责主要包括集群、节点和索引的管理，不负责文档级别的管理；data节点可以关闭http功能。

20.1.1.1.1.11. Elasticsearch是如何避免脑裂现象的

当集群中master候选的个数不小于3个（node.master: true）。可以通过discovery.zen.minimum_master_nodes 这个参数的设置来避免脑裂，设置为(N/2)+1。

这里node.master : true 是说明你是有资格成为master，并不是指你就是master。是皇子，不是皇帝。假如有10个皇子，这里应该设置为（10/2）+1=6，这6个皇子合谋做决策，选出新的皇帝。另外的4个皇子，即使他们全聚一起也才四个人，不足合谋的最低人数限制，他们不能选出新皇帝。

假如discovery.zen.minimum_master_nodes 设置的个数为5，有恰好有10个master备选节点，会出现什么情况呢？5个皇子组成一波，选一个皇帝出来，另外5个皇子也够了人数限制，他们也能选出一个皇帝来。此时一个天下两个皇帝，在es中就是脑裂。

假如集群master候选节点为2的时候，这种情况是不合理的，最好把另外一个node.master改成false。如果我们不改节点设置，还是套上面的(N/2)+1公式，此时discovery.zen.minimum_master_nodes应该设置为2。这就出现一个问题，两个master备选节点，只要有一个挂，就选不出master了。

我还是用皇子的例子来说明。假如先皇在位的时候规定，必须他的两个皇子都在的时候，才能从中2选1 继承皇位。万一有个皇子出意外挂掉了，就剩下一个皇子，天下不就没有新皇帝了么。

20.1.1.1.1.12. 编码转换解决中文乱码

方案1：input中的codec=>plain转码：将GB2312的文本编码，转为UTF-8的编码。

```
codec => plain {
 charset => "GB2312"
}
```

方案2：在filebeat中实现编码的转换

```
filebeat.prospectors:
- input_type: log
 paths:
 - /data/log/performanceTrace.txt
 encoding: GB2312
```

21. Jenkins与Gitlab实现企业级CI/CD

21.1.1.1.1.1. jenkins是什么，为什么要用它

Jenkins是一个开源的、可扩展的持续集成、交付、部署（软件/代码的编译、打包、部署）的基于web界面的平台。允许持续集成和持续交付项目，无论用的是什么平台，可以处理任何类型的构建或持续集成。

Jenkins是一种使用Java编程语言编写的开源持续集成软件工具，用于实时测试和报告较大代码库中的孤立更改。 Jenkins软件使开发人员能够快速找到并解决代码库中的缺陷，并自动进行构建测试。

21.1.1.1.1.2. CI/CD是什么

CI(Continuous integration，中文意思是持续集成)是一种软件开发时间。持续集成强调开发人员提交了新代码之后，立刻进行构建、（单元）测试。根据测试结果，我们可以确定新代码和原有代码能否正确地集成在一起。

CD(Continuous Delivery， 中文意思持续交付)是在持续集成的基础上，将集成后的代码部署到更贴近真实运行环境(类生产环境)中。比如，我们完成单元测试后，可以把代码部署到连接数据库的Staging环境中更多的测试。如果代码没有问题，可以继续手动部署到生产环境。

21.1.1.1.1.3. 什么是Jenkins pipeline

Pipeline，简而言之，就是一台运行于Jenkins上的工作流框架，将原本独立运行于单个或者多个节点的任务连接起来，实现单个任务难以完成的复杂流程编排与可视化。
Jenkins Pipeline是一组插件，让Jenkins可以实现持续交付管道的落地和实施。

21.1.1.1.1.4. 什么是Blue Ocean

Blue Ocean是pipeline的可视化UI。同时他兼容经典的自由模式的job。Jenkins Pipeline从头开始设计，但仍与自由式作业兼容，Blue Ocean减少了经典模式下的混乱并为团队中的每个成员增加了清晰度。
Blue Ocean的主要特点包括：

连续交付（CD）管道的复杂可视化，可以让您快速直观地理解管道状态。
管道编辑器 - 通过引导用户通过直观和可视化的过程来创建管道，从而使管道的创建变得平易近人。
个性化以适应团队中每个成员的基于角色的需求。
在需要干预和/或出现问题时确定精确度。Blue Ocean显示的标注了关键步骤，促进异常处理和提高生产力。

21.1.1.1.1.5. Jenkins支持哪些SCM工具

Jenkins支持版本控制工具，包括AccuRev，CVS，Subversion，Git，Mercurial，Perforce，ClearCase和RTC，并且可以执行基于Apache Ant，Apache Maven和sbt的项目以及任意的shell脚本和Windows批处理命令。

21.1.1.1.1.6. 如何在Jenkins中计划构建?

在Jenkins中，在工作配置下，我们可以定义各种构建触发器。只需找到“构建触发器”部分，然后选中“定期构建”复选框即可。 使用定期构建，您可以按星期几或星期几以及执行构建的时间安排构建定义。 “时间表”文本框的格式如下： MINUTE（0-59），HOUR （0-23），DAY（1-31），MONTH（1-12），WEEK （0-7）

21.1.1.1.1.7. 如何在Jenkins中创建多分支管道?

Multibranch Pipeline项目类型使您可以为同一项目的不同分支实现不同的Jenkinsfile。 在Multibranch Pipeline项目中，Jenkins自动发现，管理和执行针对在源代码管理中包含Jenkinsfile的分支的管道。

21.1.1.1.1.8. 如何在Jenkins中配置自动构建?

在Jenkins中的构建可以定期触发（按计划，在配置中指定），或者在检测到项目中的源更改时触发，或者可以通过请求URL自动触发： http://YOURHOST/jenkins/job/PROJECTNAME/build

21.1.1.1.1.9. 常见的git前端管理工具有哪些

gitlab：
特点是一个人维系一个分支。 使用方便灵活
每一次merge作为一次REVIEW，每次merge可以包含多个commit

gerrit：
特点是一个团队维系一个分支（业务），具有更细力度的权限管理方案，
每一次commit作为一次REVIEW，因存在changeID所以可以反复REVIEW

21.1.1.1.1.10. Jenkins服务器如何迁移

只需复制相应的 job 目录，即可将 job 从一个 Jenkins 服务器移动到另一个。
通过使用其它名称克隆 job 目录来制作现有 job 的副本。
通过重命名目录来重命名现有 job。请注意，如果你更改了 job 名称，则需要更改尝试调用该重命名 job 的所有 job 。

21.1.1.1.1.11. 常见的代码部署方式

蓝绿部署：指的是不停老版本代码(不影响上一个版本访问)，而是在另外一套环境部署新版本然后进行测试，测试通
过后将用户流量切到新版本，其特点为业务无中断，升级风险相对较小。

金丝雀发布：也叫灰度发布，是指在黑与白之间，能够平滑过渡的一种发布方式，灰度发布是增量发布的一种类型，
灰度发布是在原有版本可用的情况下，同时部署一个新版本应用作为“金丝雀”(小白鼠)，测试新版本的性能和表
现，以保障整体系统稳定的情况下，尽早发现、调整问题。

滚动发布：一般是取出一个或者多个服务器停止服务，执行更新，并重新将其投入使用。周而复始，直到集群中所
有的实例都更新成新版本。

A/B测试：同时运行两个APP环境，但是蓝绿部署完全是两码事，A/B 测试是用来测试应用功能表现的方法，例
如可用性、受欢迎程度、可见性等等，蓝绿部署的目的是安全稳定地发布新版本应用，并在必要时回滚，即蓝绿部
署是一套正式环境环境在线，而A/B测试是两套正式环境在线。

21.1.1.1.1.12. sonerqube是什么

是一款用于代码质量管理的开源工具，它主要用于管理源代码的质量。 通过插件形式，可以支持众多计算机语言，比如 java，C#，go，C/C++，PL/SQL，Cobol，JavaScrip，Groovy 等。sonar可以通过PMD,CheckStyle,Findbugs等等代码规则检测工具来检测你的代码，帮助你发现代码的漏洞，Bug，坏的编程习惯等信息

22. Docker+K8s企业级实战案例全方位讲解

22.1.1.1.1.1. Docker与虚拟机有何不同

Docker不是虚拟化方法。它依赖于实际实现基于容器的虚拟化或操作系统级虚拟化的其他工具。为此，Docker最初使用LXC驱动程序，然后移动到libcontainer现在重命名为runc。Docker主要专注于在应用程序容器内自动部署应用程序。应用程序容器旨在打包和运行单个服务，而系统容器则设计为运行多个进程，如虚拟机。因此，Docker被视为容器化系统上的容器管理或应用程序部署工具。
A 容器不需要引导操作系统内核，因此可以在不到一秒的时间内创建容器。此功能使基于容器的虚拟化比其他虚拟化方法更加独特和可取。
B 由于基于容器的虚拟化为主机增加了很少或没有开销，因此基于容器的虚拟化具有接近本机的性能。
C 对于基于容器的虚拟化，与其他虚拟化不同，不需要其他软件。
D 主机上的所有容器共享主机的调度程序，从而节省了额外资源的需求。
E 与虚拟机映像相比，容器状态（Docker或LXC映像）的大小很小，因此容器映像很容易分发。
F 容器中的资源管理是通过cgroup实现的。Cgroups不允许容器消耗比分配给它们更多的资源。虽然主机的所有资源都在虚拟机中可见，但无法使用。这可以通过在容器和主机上同时运行top或htop来实现。所有环境的输出看起来都很相似。

22.1.1.1.1.2. 什么是Docker Swarm

Docker Swarm是Docker的本机群集。它将Docker主机池转变为单个虚拟Docker主机。Docker Swarm提供标准的Docker API，任何已经与Docker守护进程通信的工具都可以使用Swarm透明地扩展到多个主机。

22.1.1.1.1.3. 如何在生产中监控Docker

Docker提供docker stats和docker事件等工具来监控生产中的Docker。我们可以使用这些命令获取重要统计数据的报告。
Docker统计数据：当我们使用容器ID调用docker stats时，我们获得容器的CPU，内存使用情况等。它类似于Linux中的top命令。
Docker事件：Docker事件是一个命令，用于查看Docker守护程序中正在进行的活动流。
一些常见的Docker事件是：attach，commit，die，detach，rename，destroy等。我们还可以使用各种选项来限制或过滤我们感兴趣的事件

22.1.1.1.1.4. 如何批量清理临时镜像文件?

```
sudo docker rmi $(sudo docker images -q -f danging=true)
```

22.1.1.1.1.5. 本地的镜像文件都存放在哪里?

于Docker相关的本地资源存放在 /var/lib/docker/目录下、其中container目录存放容器信息、graph目录存放镜像信息、aufs目录存放具体的镜像底层文件

22.1.1.1.1.6. 构建Docker镜像应该遵循哪些原则?

整体原则上、尽量保持镜像功能的明确和内容的精简、要点包括:
1、尽量选取满足需求但较小的基础系统镜像、建议选择Debian:wheezy镜像，仅有86MB大小
2、清理编译生成的文件、安装包的缓存等临时文件
3、安装各个软件时候要指定准确的版本号、并避免引入不需要的依赖
4、从安全的角度考虑、应用尽量使用系统的库和依赖
5、使用Dockerfile创建镜像时候要添加dockerignore文件或使用干净的工具目录

22.1.1.1.1.7. 容器退出后、通过docker ps 命令查不到数据丢失么?

容器退出后会处于中止(exited)状态、此时可以通过 docker ps -a 查看、其中数据不会丢失、还可以通过docker start来启动、只有删除容器才会清除数据

22.1.1.1.1.8. 如何临时退出一个正在交付的容器的终端、而不中止它?

按Ctrl+p 后按Ctrl+q、如果按Ctrl+c 会使容器内的应用进程终止、进而会使容器终止

22.1.1.1.1.9. 很多应用容器都是默认后台运行的、怎么查看它们的输出和日志信息?

sudo docker logs 、后面容器名称或容器ID号
sudo docker logs -f -t --tail 100 容器名称或容器ID号

22.1.1.1.1.10. 可以在一个容器中同时运行多个应用进程吗?

一般不推荐在同一个容器内运行多个应用进程、如果有类似需求、可以通过额外的进程管理机制、比如supervisord来管理所运行的进程

22.1.1.1.1.11. 如何控制容器占用系统资源(cpu、内存)的份额?

在使用docker create命令创建容器或使用docker run 创建并运行的时候、可以使用 -c|-cpu-shares[=0]参数来调整同期使用cpu的权重、使用-m|-memory参数来调整容器使用内存的大小

22.1.1.1.1.12. Kubernetes与Docker有什么关系?

Docker提供容器的生命周期管理，Docker镜像构建运行时容器。但是，由于这些单独的容器必须通信，因此使用Kubernetes。因此，我们说Docker构建容器，这些容器通过Kubernetes相互通信。因此，可以使用Kubernetes手动关联和编排在多个主机上运行的容器。

22.1.1.1.1.13. 什么是 Etcd?

Etcd 是用 Go 编程语言编写的一个分布式键值存储，用于协调分布式工作的软件。因此，Etcd 用来存储 Kubernetes 集群的配置数据，这些数据代表在任何给定时间点的集群状态。

22.1.1.1.1.14. 五种Pod共享资源

Pod是K8s最基本的操作单元，包含一个或多个紧密相关的容器，一个Pod可以被一个容器化的环境看作应用层的“逻辑宿主机”；一个Pod中的多个容器应用通常是紧密耦合的，Pod在Node上被创建、启动或者销毁；每个Pod里运行着一个特殊的被称之为Volume挂载卷，因此他们之间通信和数据交换更为高效，在设计时我们可以充分利用这一特性将一组密切相关的服务进程放入同一个Pod中。
同一个Pod里的容器之间仅需通过localhost就能互相通信。一个Pod中的应用容器共享五种资源：
PID命名空间：Pod中的不同应用程序可以看到其他应用程序的进程ID。
网络命名空间：Pod中的多个容器能够访问同一个IP和端口范围。
IPC命名空间：Pod中的多个容器能够使用SystemV IPC或POSIX消息队列进行通信。
UTS命名空间：Pod中的多个容器共享一个主机名。
Volumes(共享存储卷)：Pod中的各个容器可以访问在Pod级别定义的Volumes。
Pod的生命周期通过Replication Controller来管理；通过模板进行定义，然后分配到一个Node上运行，在Pod所包含容器运行结束后，Pod结束。
Kubernetes为Pod设计了一套独特的网络配置，包括：为每个Pod分配一个IP地址，使用Pod名作为荣期间通信的主机名等

22.1.1.1.1.15. Replica Set 和 Replication Controller之间有什么区别?

Replica Set 和 Replication Controller几乎完全相同。它们都确保在任何给定时间运行指定数量的pod副本。不同之处在于复制pod使用的选择器。Replica Set使用基于集合的选择器，而Replication Controller使用基于权限的选择器。
Equity-Based选择器：这种类型的选择器允许按标签键和值进行过滤。因此，在外行术语中，基于Equity的选择器将仅查找与标签具有完全相同短语的pod。
示例：假设您的标签键表示app = nginx，那么，使用此选择器，您只能查找标签应用程序等于nginx的那些pod。
Selector-Based选择器：此类型的选择器允许根据一组值过滤键。因此，换句话说，基于Selector的选择器将查找已在集合中提及其标签的pod。
示例：假设您的标签键在(nginx，NPS，Apache)中显示应用程序。然后，使用此选择器，如果您的应用程序等于任何nginx，NPS或Apache，则选择器将其视为真实结果。

22.1.1.1.1.16. 简要说下Kubernetes有哪些核心组件以及这些组件负责什么工作?

etcd：提供数据库服务保存了整个集群的状态
kube-apiserver：提供了资源操作的唯一入口，并提供认证、授权、访问控制、API注册和发现等机制
kube-controller-manager：负责维护集群的状态，比如故障检测、自动扩展、滚动更新等
cloud-controller-manager：是与底层云计算服务商交互的控制器
kub-scheduler：负责资源的调度，按照预定的调度策略将Pod调度到相应的机器上
kubelet：负责维护容器的生命周期，同时也负责Volume和网络的管理
kube-proxy：负责为Service提供内部的服务发现和负载均衡，并维护网络规则
container-runtime：是负责管理运行容器的软件，比如docker

22.1.1.1.1.17. 经典pod的生命周期

od都处于以下几种状态之一，可通过查询Pod详情查看。

Pending 部署Pod事务已被集群受理，但当前容器镜像还未下载完。
Running 所有容器已被创建，并被部署到k8s节点。
Successed Pod成功退出，并不会被重启。
Failed Pod中有容器被终止。
Unknown 未知原因，如kube-apiserver无法与Pod进行通讯。
详细叙述如下：

首先拖取Pod内容器的镜像，选择某个Node节点并启动Pod。 监控Pod状态，若Pod终止则根据策略决定是否重新调度。 Pod退出，并根据策略决定是否重启。

22.1.1.1.1.18. 详述kube-proxy原理，一个请求是如何经过层层转发落到某个pod上的整个过程。请求可能来自pod也可能来自外部。

kube-proxy部署在每个Node节点上，通过监听集群状态变更，并对本机iptables做修改，从而实现网络路由。 而其中的负载均衡，也是通过iptables的特性实现的。

另外我们需要了解k8s中的网络配置类型，有如下几种：

hostNetwork Pod使用宿主机上的网络，此时可能端口冲突。
hostPort 宿主机上的端口与Pod的目标端口映射。
NodePort 通过Service访问Pod，并给Service分配一个ClusterIP。

22.1.1.1.1.19. deployment/rs有什么区别。其使用方式、使用条件和原理是什么。

deployment是rs的超集，提供更多的部署功能，如：回滚、暂停和重启、 版本记录、事件和状态查看、滚动升级和替换升级。
如果能使用deployment，则不应再使用rc和rs。

22.1.1.1.1.20. rc/rs实现原理，详述从API接收到一个创建rc/rs的请求，到最终在节点上创建pod的全过程，尽可能详细。另外，当一个pod失效时，kubernetes是如何发现并重启另一个pod的?

Replication Controller 可以保证Pod始终处于规定的副本数。
而当前推荐的做法是使用Deployment+ReplicaSet。
ReplicaSet 号称下一代的 Replication Controller，当前唯一区别是RS支持set-based selector。
RC是通过ReplicationManager监控RC和RC内Pod的状态，从而增删Pod，以实现维持特定副本数的功能。
RS也是大致相同。

22.1.1.1.1.21. 查看ops这个命名空间下的所有pod，并显示pod的IP地址

```
kubectl get pods -n ops -o wide
```

22.1.1.1.1.22. 查看atest命名空间下的pod名称为atest-12ddde-fdfde的日志，并显示最后30行

```
kubectl logs atest-12ddde-fdfde -n atest | tail -n 30
```

22.1.1.1.1.23. 列出所有 namespace 中的所有 pod

```
kubectl get pods --all-namespaces
```

22.1.1.1.1.24. 如何删除test命名空间下某个deploymemt，名称为gerrit

```
kubectl delete deploy gerrit -n test
```

22.1.1.1.1.25. 如何缩减test命名空间下deployment名称为gitlab的副本数为1

```
kubectl scale deployment gitlab -n test --replicas=1
```

22.1.1.1.1.26. 如何设置节点test-node-01为不可调度以及如何取消不可调度

```
kubectl cordon test-node-01     #设置test-node-10为不可调度
kubectl uncordon test-node-01   #取消
```

22.1.1.1.1.27. 集群如何预防雪崩

```
1、为每个pod设置资源限制
2、设置Kubelet资源预留
```

22.1.1.1.1.28. controller-manager 与etcd进行 通信吗?

```
不通信
原因：controller-manager 和 scheduler 它们都是和 kube-apiserver通信，然后 kube-
apiserver 再和etcd通信。
```

22.1.1.1.1.29. k8s集群节点需要关机维护，需要怎么操作

```
# 驱逐 node 节点上 pod
$ kubectl drain k8s-node-01 --force --ignore-daemonsets

# 关机
$ init 0
```

22.1.1.1.1.30. headless-service使用场景

```
自主选择权 client可以自主选择哪个server
Headless Service 的对应的每一个 Endpoints，即每一个Pod，都会有对应的DNS域名，这样Pod之间就
可以互相访问。
```