1. 常见的 MySQL 架构?

```
主从机构
主主架构
主主主架构
```

2. mysql 复制基本原理流程

```
1. 主：binlog 线程——记录下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。
```

3. MySQL 复制的线程有几个及之间的关联

```
MySQL 的复制是基于如下 3 个线程的交互：
1. Master 上面的 binlog dump 线程，该线程负责将 master 的 binlog event 传到 slave；
2. Slave 上面的 IO 线程，该线程负责接收 Master 传过来的 binlog，并写入 relay log；
3. Slave 上面的 SQL 线程，该线程负责读取 relay log 并执行；
```

4. 如何检测出 mysql 主从有哪些数据不一致？如何修复?

```
1. 可以使用 Percana-Toolkit 工具组件的 pt-table-checksum 校验出来
2. 然后在用 pt-check-sync 进行同步更新（根据实际情况使用）
```

5. 线上 MySQL 主从复制故障如何解决?

```
登陆从库
1. 执行 stop slave; 停止主从同步
2. 然后 set global sql_slave_skip_counter = 1; 跳过一步错误
3. 最后执行 start slave; 并查看主从同步状态

需要重新进行主从同步操作步骤如下
进入主库
1. 进行全备数据库并刷新 binlog，查看主库此的状态
2. 恢复全备文件到从库，然后执行 change master
3. 开启主从同步 start slave; 并查看主从同步状态
```

6. 如何监控主从复制是否故障？（或者 zabbix 监控 mysql 主从复制状态是否正常的脚本思路是？）

```
mysql -uroot -ppassowrd -e "show slave status\G" | grep -E
"Slave_IO_Running|Slave_SQL_Running" | awk '{print $2}' | grep -c Yes
通过判断 Yes 的个数来监控主从复制状态，正常情况等于 2
```

7. 生产一主多从从库宕机，如何手工恢复?

```
1. 执行 stop slave 或者停止服务
2. 修复好从库数据库
3. 然后重新操作主库同步
```

8. 生产一主多从主库宕机，如何手工恢复?

```
1. 登陆各个从库停止同步，并查看谁的数据最新，将它设置为新主库让其它从库同步其数据
2. 修复好主库之后，生新操作主从同步的步骤就可以了

# 需要注意的新的主库如果之前是只读，需要关闭此功能让其可写
# 需要在新从库创建与之前主库相同的同步的用户与权限
# 其它从库执行 change master to master_port=新主库的端口，start slave
```

9. MySQL 出现复制延迟有哪些原因？如何解决?

```
1. 需要同步的从库数据太多
2. 从库的硬件资源较差，需要提升
3. 网络问题，需要提升网络带宽
4. 主库的数据写入量较大，需要优配置和硬件资源
5. sql 语句执行过长导致，需要优化
6. 版本 5.6 之前是单线程复制，之后是多线程，也可采用 5.7 以上版本
```

10. 给出企业生产大型 MySQL 集群架构可行备份方案?

```
1. 双主多从，主从同步的架构，然后实行某个从库专业做为备份服务器
2. 编写脚本实行分库分表进行备份，并加入定时任务
3. 最终将备份服务推送至内网专业服务器，数据库服务器本地保留一周
4. 备份服务器根据实际情况来保留备份数据（一般 30 天）
```

11. 讲述一下 Tomcat 8005、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. Tomcat 有三种工作模式：Bio、Nio 和 Apr，他们工作原理是?

```
Bio(Blocking I/O)：默认工作模式，阻塞式 I/O 操作，没有任何优化技术处理，性能比较低。
Nio(New I/O or Non-Blocking)：非阻塞式 I/O 操作，有比 Bio 有更好的并发处理性能。
Apr(Apache Portable Runtime，Apache 可移植运行库)：首选工作模式，主要为上层的应用程序提供一个
可以跨越多操作系统平台使用的底层支持接口库。
```

tomcat 利用基于 Apr 库 tomcat-native 来实现操作系统级别控制，提供一种优化技术和非阻塞式 I/O 操作，大大提高并发处理能力。但是需要安装 apr 和 tomcat-native 库。

13. 请解释 Tomcat 中使用的连接器是什么？

在 Tomcat 中，使用了两种类型的连接器：
HTTP 连接器 ：它有许多可以更改的属性，以确定它的工作方式和访问功能，如重定向和代理转发
AJP 连接器 ：它以与 HTTP 连接器相同的方式工作，但是他们使用的是 HTTP 的 AJP 协议。AJP 连接器通常通过插件技术 mod_jk 在 Tomcat 中实现

14. Tomcat 调优大概有哪些思路?

1. 增加最大连接数
2. 调整工作模式
3. 启用 gzip 压缩
4. 调整 JVM 内存大小
5. 与 Apache 或 Nginx 整合，实现动静分离
6. 合理选择垃圾回收算法
7. 尽量使用较新 JDK 版本

15. 请写下命令检查 nginx 的当前配置文件，然后平滑重启

```
1. 确认 nginx 配置文件的语法是否正确
nginx -t

2. 平滑重启
kill -HUP nginx 进程号
kill -HUP /路径/nginx.pid
```

16. 请简单描述 nginx 与 php-fpm 的两种连接方式及其优缺点

在 linux 中，nginx 服务器和 php-fpm 可以通过 tcp socket 和 unix socket 两种方式实现。
unix socket 是一种终端，可以使同一台操作系统上的两个或多个进程进行数据通信。这种方式需要在 nginx 配置文件中填写 php-fpm 的 pid 文件位置，效率要比 tcp socket 高。
tcp socket 这种通信方式，需要在 nginx 配置文件中填写 php-fpm 运行的 ip 地址和端口号。这种方式的优点是可以跨服务器，当 nginx 和 php-fpm 不在同一台机器上时，只能使用这种方式。

17. 你常用的 Nginx 模块，用来做什么

```
rewrite 模块：实现重写功能
access 模块：来源控制
ssl 模块：安全加密
ngx_http_gzip_module：网络传输压缩模块
ngx_http_proxy_module：模块实现代理
```

```
ngx_http_upstream_module：模块实现定义后端服务器列表
ngx_cache_purge：实现缓存清除功能
```

18. 指出 Nginx 支持哪几种负载均衡模式，并指出各模式的应用场景

```
1. roundrobin 轮询方式，依次将请求分配到各个后台服务器中，默认的负载均衡方式，适用于后台机器性能一致的情况， 挂掉的机器可以自动从服务列表中剔除。
2. weight 根据权重来分发请求到不同的机器中，适用于后台机器性能不一样的情况。
3. ip_hash 根据请求者 ip 的 hash 值将请求发送到后台服务器中，可以保证来自同一 ip 的请求被打到固定的机器上，可以解决 session 问题。
4. url_hash 根据请求的 url 的 hash 值将请求分到不同的机器中，当后台服务器为缓存的时候效率高。
5. fair 根据后台响应时间来分发请求，响应时间短的分发的请求多。
```

19. 讲一下 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
```

20. keepalived 的主要模块有哪些?

```
keepalived 主要有三个模块，分别是 core、check 和 vrrp。
core 模块为 keepalived 的核心，负责主进程的启动、维护及全局配置文件的加载和解析。
check 负责健康检查，包括常见的各种检查方式
vrrp 模块是来实现 VRRP 协议的
```

21. keepalived 如何做到健康检查?

```
Keepalived 健康检查方式配置
HTTP_GET|SSL_GET
HTTP_GET | SSL_GET
{
    url {
        path /
        # HTTP/SSL 检查的 url 可以是多个
        digest <STRING>
        # HTTP/SSL 检查后的摘要信息用工具 genhash 生成
        status_code 200 # HTTP/SSL 检查返回的状态码
    }
    connect_port 80
    # 连接端口
    bindto<IPADD>
    connect_timeout 3
    # 连接超时时间
    nb_get_retry 3
    # 重连次数
    delay_before_retry 2
```

```
    #连接间隔时间
}
```

22. Keepalived 怎么实现高可用?

Keepalived 高可用服务对之间的故障切换转移，是通过 VRRP 协议来实现的。
在 Keepalived 服务正常工作时，主 Master 节点会不断地向备节点发送（多播的方式）心跳消息，用以告诉备 Backup 节点自己还活着。
当主 Master 节点发生故障时，就无法发送心跳消息，备节点也就因此无法继续检测到来自主 Master 节点的心跳了，于是调用自身的接管程序，接管主 Master 节点的 IP 资源及服务。
而当主 Master 节点恢复时，备 Backup 节点又会释放主节点故障时自身接管的 IP 资源及服务，恢复到原来的备用角色。

23. Linux 集群主要有哪几类?

1. LB 负载均衡集群
负载均衡集群主要是提高服务的响应能力，如果一组计算机节点（或者一组进程）提供相同的（同质的）服务，那么对服务的请求就应该均匀的分摊到这些节点上。软件级的比较常见的有如下两种：LVS、Haproxy

2. HA 高可用性集群
高可用性集群主要是提供 7*24 小时不间断服务的,如果某台宕机了，会自动的切换到其他计算机上面工作，从而达到高可用的效果。
HA 高可用集群的解决方案常见的有以下几种
heartbeat、corosync+openais、RHCS、ultramokey、keepalived

3. HP 高性能集群
高性能集群主要是用于需要大量 CPU 运算的场景中，比如说天气预报，国外 3D 大片的特效制作，等等一系列需要做大量运算
的应用，HP 高性能集群的解决方案常见的有 bowerful

24. 常见几种负载均衡方式的比较?

一、LVS 的特点
1. 工作在网络 4 层上，抗负载能力强，作分发之用;
2. 配置性比较低;
3. 工作稳定，自身具备的双机热备方案;
4. 应用范围比较广，可以对所有应用做负载均衡;

二、NGINX 的特点
1. 工作在网络的 7 层之上;
2. 对网络的依赖比较小;
3. 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;
5. 可以通过端口检测到服务器内部的故障,
6. 对请求的异步处理可以帮助节点服务器减轻负载;
7. 能支持 http 和 Email;
8. 默认的只有 Round-robin 和 IP-hash 两种负载均衡算法;

三、Haproxy 的特点
1. 工作在网络 7 层之上。

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 检测后端的服务器出问题的检测
4. 更多的负载均衡策略
5. 有更出色的负载均衡速度
6. HAProxy 可以对 Mysql 进行负载均衡，对后端的 DB 节点进行检测和负载均衡

25. lvs 调度算法中的最小连接数算法原理是什么

最少连接调度算法是把新的连接请求分配到当前连接数最小的服务器，最小连接调度是一种动态调度短算法，它通过服务器当前所活跃的连接数来估计服务器的负载均衡，调度器需要记录各个服务器已建立连接的数目
当一个请求被调度到某台服务器，其连接数加 1，当连接中止或超时，其连接数减 1，在系统实现时， 我们也引入当服务器的权值为 0 时，表示该服务器不可用而不被调度。

26. 简单介绍 lvs 的三种工作模型

1. NAT 模型
NAT 模型是通过网络地址转换来实现的,工作方式是，首先用户请求到达前端的负载均衡器，然后负载均衡器根据事先定义好的调度算法将用户请求的目标地址修改为后端的应用服务器，应用程序服务器处理好请求之后将结果返回给用户，期间必须要经过负载均衡器，负载均衡器将报文的源地址 改为用户请求的目标地址，再转发给用户，从而完成整个负载均衡的过
程。

2. DR 模型
DR 模型是通过路由技术实现的负载均衡技术，这种模型与NAT模型不同的地方是，负载均衡器通过改写用户请求报文中的 MAC 地址，将请求发送到 Real Server，而 Real Server 直接响应用户，这样就大大的减少负载均衡器的压力，DR 模型也是
用的最多的一种。

3. TUN模型
TUN 模型是通过 IP 隧道技术实现的，TUN 模型跟 DR 模型有点类似，不同的地方是 负载均衡器(Director Server) 跟 应用服务器(Real Server) 通信的机制是通过 IP 隧道技术将用户的请求转发到某个 Real Server，而 Real Server 也是直接响应用户的。

27. 集群中资源隔离的解决方案?

1. 当集群分裂成两个小集群时会发生资源争用的情况，为避免争用后端存储系统而造成灾难性的系统崩溃，集群系统引入了投票机制，只有拥有半数以上合法票数的集群才能存活，否则就推出集群系统。
2. 当集群为偶数时，如果分裂，两边可能都掌握相等的票数；因此，集群系统不应该为偶数，如果是偶数则需要一个额外的 ping 节点参与投票。
3. 票数不足的集群退出集群服务后，为了保证它不会争用资源需要 STONITH 机制来进行资源隔离。