1、RPC 原理
（1）为什么会出现 RPC?
RPC(Remote Procedure Call Protocol)——远程过程调用协议。

一般来说，自己写程序然后本地调用，这种程序的特点是服务的消费方和提供方。当我们进入公司时，面对的很可能就是成千上万的服务提供方，这时候就需要使用 RPC 来进行远程服务调用。RPC 将原来的本地调用转变为调用远端的服务器上的方法，给系统的处理能力和吞吐量带来了近似于无限制提升的可能。

（2）RPC 的组成
①客户端：服务的调用方
②客户端存根：存放服务端的地址消息，再将客户端的请求参数打包成网络消息，③然后通过网络远程发送给服务方。
④服务端：真正的服务提供者。
⑤服务端存根：接收客户端发送过来的消息，将消息解包，并调用本地的方法。

2、RPC 的过程?
3、如何做到透明化远程服务调用?
动态代理，把本地调用代理成网络调用

4、如何进行服务发布？（Zookeeper）
5、如何进行序列化与反序列化？（Protobuf、Thrift、Avro）
6、如何进行通信？（NIO--->Netty）
7、HashMap 原理
8、Redis 缓存回收机制
（1）数据过期:
①定时删除策略：Redis 启动一个定时器监控所有的 key,一旦有过期的话就进行删除（遍历所有 key,非常耗费 CPU）
②惰性删除策略：获取 key 的时候判断是否过期， 过期则进行删除
Redis 采用的方式:①(随机抓取一部分 key 进行检测)+②

（2）内存淘汰:
①noeviction：当内存不足以容纳新写入数据时，新写入操作会报错。(Redis 默认策略)
②allkeys-lru：当内存不足以容纳新写入数据时，在键空间中，移除最近最少使用的 Key。（LRU 推荐使用）
③allkeys-random：当内存不足以容纳新写入数据时，在键空间中，随机移除某个 Key。
④volatile-lru：当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，移除最近最少使用的 Key。这种情况一般是把 Redis 既当缓存，又做持久化存储的时候才用。
⑤volatile-random：当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，随机移除某个 Key。
⑥volatile-ttl：当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，有更早过期时间的 Key 优先移除。不推荐。如果没有对应的键，则回退到 noeviction 策略。

9、Redis 主从同步

（1）主从复制作用
①数据冗余
②故障恢复（服务冗余）
③负载均衡
④读写分离（主节点写操作、从节点读操作）

（2）主从复制过程
①连接建立阶段
步骤 1：保存主节点信息
步骤 2：建立 socket 连接
步骤 3：发送 ping 命令
步骤 4：身份验证
步骤 5：发送从节点端口信息
②数据同步阶段
从节点向主节点发送 psync 命令
根据主从节点当前状态的不同，可以分为全量复制和部分复制
③命令传播阶段
主从节点进入命令传播阶段；在这个阶段主节点将自己执行的写命令发送给从节点，从节点接收命令并执行，从而保证主从节点数据的一致性。

（3）介绍全量复制和部分复制
①全量复制：用于初次复制或其他无法进行部分复制的情况，将主节点中的所有数据都发送给从节点，是一个非常重型的操作。
②部分复制：用于网络中断等情况后的复制，只将中断期间主节点执行的写命令发送给从节点，与全量复制相比更加高效。需要注意的是，如果网络中断时间过长，导致主节点没有能够完整地保存中断期间执行的写命令，则无法进行部分复制，仍使用全量复制。

（4）主从复制缺点：故障恢复无法自动化；写操作无法负载均衡；存储能力受到单机的限制。

10、为什么会有哨兵机制？
在主从复制的基础上，哨兵实现了自动化的故障恢复。

11、哨兵机制作用？
（1）监控（Monitoring）：哨兵会不断地检查主节点和从节点是否运作正常。
（2）自动故障转移（Automatic failover）：当主节点不能正常工作时，哨兵会开始自动故障转移操作，它会将失效主节点的其中一个从节点升级为新的主节点，并让其他从节点改为复制新的主节点。
（3）配置提供者（Configuration provider）：客户端在初始化时，通过连接哨兵来获得当前 Redis 服务的主节点地址。
（4）通知（Notification）：哨兵可以将故障转移的结果发送给客户端。

12、哨兵机制节点组成？
它由两部分组成，哨兵节点和数据节点：

（1）哨兵节点：哨兵系统由一个或多个哨兵节点组成，哨兵节点是特殊的 redis 节点，不存储数据。
（2）数据节点：主节点和从节点都是数据节点。

13、哨兵机制原理?
（1）定时任务：每个哨兵节点维护了 3 个定时任务。定时任务的功能分别如下：通过向主从节点发送 info 命令获取最新的主从结构；通过发布订阅功能获取其他哨兵节点的信息；通过向其他节点发送 ping 命令进行心跳检测，判断是否下线。
（2）主观下线：在心跳检测的定时任务中，如果其他节点超过一定时间没有回复，哨兵节点就会将其进行主观下线。顾名思义，主观下线的意思是一个哨兵节点“主观地”判断下线；与主观下线相对应的是客观下线。
（3）客观下线：哨兵节点在对主节点进行主观下线后，会通过 sentinel is-master-down-by-addr 命令询问其他哨兵节点该主节点的状态；如果判断主节点下线的哨兵数量达到一定数值，则对该主节点进行客观下线。
（4）选举领导者哨兵节点：当主节点被判断客观下线以后，各个哨兵节点会进行协商，选举出一个领导者哨兵节点，并由该领导者节点对其进行故障转移操作。监视该主节点的所有哨兵都有可能被选为领导者，选举使用的算法是 Raft 算法；Raft 算法的基本思路是先到先得：即在一轮选举中，哨兵 A 向 B 发送成为领导者的申请，如果 B 没有同意过其他哨兵，则会同意 A 成为领导者。选举的具体过程这里不做详细描述，一般来说，哨兵选择的过程很快，谁先完成客观下线，一般就能成为领导者。
（5）故障转移：选举出的领导者哨兵，开始进行故障转移操作，该操作大体可以分为 3 个步骤：

①在从节点中选择新的主节点：选择的原则是，首先过滤掉不健康的从节点；然后选择优先级最高的从节点(由 slave-priority 指定)；如果优先级无法区分，则选择复制偏移量最大的从节点；如果仍无法区分，则选择 runid 最小的从节点。
②更新主从状态：通过 slaveof no one 命令，让选出来的从节点成为主节点；并通过 slaveof 命令让其他节点成为其从节点。
③将已经下线的主节点(即 6379)设置为新的主节点的从节点，当 6379 重新上线后，它会成为新的主节点的从节点。

14、哨兵机制缺点
写操作无法负载均衡；存储能力受到单机的限制。（Redis 集群解决了该情况）

15、Zookeeper 锁是如何实现的?
一般使用 Curator 进行使用 Zookeeper 锁，例如有两个客户端 A 和客户端 B,首先 A 先在锁节点下创建例如 01 子节点的锁，然后再获取节点信息，发现自己的 01 节点排名第一，那么就获得锁。
客户端 B 也需要获取锁，现在锁节点下创建例如 02 的子节点，然后再获取锁节点信息，发现锁节点信息为[01,02],并不排第一，因此获取不到锁，客户端 B 会在他的顺序节点的上一个顺序节点加一个监听器。
当客户端 A 使用完锁，删除 01 节点，客户端 B 获取到 01 删除的监听，然后发现自己的 02 节点排名第一，那么就获取到锁。

16、分布式缓存读写不一致问题
17、Java 线程你是怎么使用的
18、数据库是如何调优的?
（1）数据表加合适的索引
（2）针对执行计划进行优化
（3）根据慢 sql 进行优化
（4）加缓存
（5）参数调优

19、git rebase 命令发生了什么?
rebase 命令可以帮我们把整个提交历史变成干净清晰的一条线。

20、手撕代码。牛客题霸上的原题，可以去看看：NC13 二叉树的最大深度