**Redis 有哪些功能：**

数据缓存功能

分布式锁的功能

支持数据持久化

支持事务

支持消息队列

**Redis 和 memcache 有什么区别：**

存储方式不同：memcache 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，这样能保证数据的持久性。

数据支持类型：memcache 对数据类型支持相对简单；Redis 有复杂的数据类型。

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 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。

value 值大小不同：Redis 最大可以达到 1gb；memcache 只有 1mb。

**redis有提供了哪些存储数据结构，redis持久化，相比于如memcached,mongodb有什么优势：**https://blog.csdn.net/TiaoZhanJi\_Xian/article/details/80301976

redis优势：支持的数据类型更丰富，速度更快(非阻塞IO多路复用模型)，支持数据持久化，支持更大的缓存(redis可达到1GB)

redis支持5种数据类型：

String、hash、list、set、sorted set(有序集合)https://www.cnblogs.com/tdws/p/5754706.html

RDB（Redis Database）持久化：指定时间间隔使用save或者bgsave命令 dump整个数据库到硬盘的一个rbd文件。

AOF（Append Only File）持久化：使用write命令记录执行的每一条指令到appendonly.aof日志文件。

混合式（4.0版本支持）https://blog.csdn.net/king624498030/article/details/78689314  
**Redis高效的原因，使用场景：**

1. 完全基于内存，大多数请求都是内存操作，非常快速；
2. 数据结构简单，操作简单；
3. 采用单线程，避免了不必要的上下文切换和竞争条件，不存在多进程或者多线程的切换，不用考虑锁带来的性能消耗；
4. 使用多路 I/O复用模型，非阻塞 IO

**Redis 为什么是单线程的：**

因为 cpu 不是 Redis 的瓶颈，Redis 的瓶颈最有可能是机器内存或者网络带宽。既然单线程容易实现，而且 cpu 又不会成为瓶颈，那就顺理成章地采用单线程的方案了。

关于 Redis 的性能，官方网站也有，普通笔记本轻松处理每秒几十万的请求。

而且单线程并不代表就慢 nginx 和 nodejs 也都是高性能单线程的代表。

**jedis 和 Redisson 有哪些区别：**

Redis支持的 Java 客户端有 Redisson、jedis、lettuce 等。

jedis：提供了比较全面的 Redis 命令的支持。

Redisson：实现了分布式和可扩展的 Java 数据结构，与 jedis 相比 Redisson 的功能相对简单，不支持排序、事务、管道、分区等 Redis 特性。

**Redis怎么保证缓存和数据库数据的一致性：**

合理设置缓存的过期时间。

新增、更改、删除数据库操作时同步更新 Redis，可以使用事物机制来保证数据的一致性。

**Redis 怎么实现分布式锁：**

Redis 分布式锁其实就是在系统里面占一个“坑”，其他程序也要占“坑”的时候，占用成功了就可以继续执行，失败了就只能放弃或稍后重试。占坑一般使用 setnx(set if not exists)指令，只允许被一个程序占有，使用完调用 del 释放锁。

缺陷：Redis 分布式锁不能解决超时的问题，分布式锁有一个超时时间，程序的执行如果超出了锁的超时时间就会出现问题。

**Redis 如何做内存优化：**

尽量使用 Redis 的散列表，把相关的信息放到散列表里面存储，而不是把每个字段单独存储，这样可以有效的减少内存使用。比如将 Web 系统的用户对象，应该放到散列表里面再整体存储到 Redis，而不是把用户的姓名、年龄、密码、邮箱等字段分别设置 key 进行存储。

**Redis 淘汰策略有哪些：**

volatile-lru：从已设置过期时间的数据集（server. db[i]. expires）中挑选最近最少使用的数据淘汰。

volatile-ttl：从已设置过期时间的数据集（server. db[i]. expires）中挑选将要过期的数据淘汰。

volatile-random：从已设置过期时间的数据集（server. db[i]. expires）中任意选择数据淘汰。

allkeys-lru：从数据集（server. db[i]. dict）中挑选最近最少使用的数据淘汰。

allkeys-random：从数据集（server. db[i]. dict）中任意选择数据淘汰。

no-enviction（驱逐）：禁止驱逐数据。

**什么是缓存穿透？怎么解决？**

缓存穿透：指查询一个一定不存在的数据，由于缓存是不命中时需要从数据库查询，查不到数据则不写入缓存，这将导致这个不存在的数据每次请求都要到数据库去查询，造成缓存穿透。

解决方案：最简单粗暴的方法如果一个查询返回的数据为空（不管是数据不存在，还是系统故障），我们就把这个空结果进行缓存，但它的过期时间会很短，最长不超过五分钟。

**Redis 常见的性能问题有哪些？该如何解决？**

主服务器写内存快照，会阻塞主线程的工作，当快照比较大时对性能影响是非常大的，会间断性暂停服务，所以主服务器最好不要写内存快照。

Redis 主从复制的性能问题，为了主从复制的速度和连接的稳定性，主从库最好在同一个局域网内。

**消息中间件mq的作用：**

多系统之间的消息通信

优点：解耦，异步，削峰

缺点：系统可用性降低：系统引入的外部依赖越多，越容易挂掉，本来你就是A系统调用BCD三个系统的接口就好了，人ABCD四个系统好好的，没啥问题，你偏加个MQ进来，万一MQ挂了咋整？MQ挂了，整套系统崩溃了，你不就完了么。

系统复杂性提高：硬生生加个MQ进来，你怎么保证消息没有重复消费？怎么处理消息丢失的情况？怎么保证消息传递的顺序性？头大头大，问题一大堆，痛苦不已

一致性问题：A系统处理完了直接返回成功了，人都以为你这个请求就成功了；但是问题是，要是BCD三个系统那里，BD两个系统写库成功了，结果C系统写库失败了，咋整？你这数据就不一致了。

**RabbitMQ 有哪些重要的角色：**

RabbitMQ 中重要的角色有：生产者、消费者和代理：

生产者：消息的创建者，负责创建和推送数据到消息服务器；

消费者：消息的接收方，用于处理数据和确认消息；

代理：就是 RabbitMQ 本身，用于扮演“快递”的角色，本身不生产消息，只是扮演“快递”的角色。

**RabbitMQ 有哪些重要的组件：**

ConnectionFactory（连接管理器）：应用程序与Rabbit之间建立连接的管理器，程序代码中使用。

Channel（信道）：消息推送使用的通道。

Exchange（交换器）：用于接受、分配消息。

Queue（队列）：用于存储生产者的消息。

RoutingKey（路由键）：用于把生成者的数据分配到交换器上。

BindingKey（绑定键）：用于把交换器的消息绑定到队列上。

**RabbitMQ 中 vhost 的作用是什么：**

vhost：每个 RabbitMQ 都能创建很多 vhost，我们称之为虚拟主机，每个虚拟主机其实都是 mini 版的RabbitMQ，它拥有自己的队列，交换器和绑定，拥有自己的权限机制。

**RabbitMQ 的消息是怎么发送的：**

首先客户端必须连接到 RabbitMQ 服务器才能发布和消费消息，客户端和 rabbit server 之间会创建一个 tcp 连接，一旦 tcp 打开并通过了认证（认证就是你发送给 rabbit 服务器的用户名和密码），你的客户端和 RabbitMQ 就创建了一条 amqp 信道（channel），信道是创建在“真实” tcp 上的虚拟连接，amqp 命令都是通过信道发送出去的，每个信道都会有一个唯一的 id，不论是发布消息，订阅队列都是通过这个信道完成的。

**RabbitMQ 怎么保证消息的稳定性：**

提供了事务的功能。

通过将 channel 设置为 confirm（确认）模式。

**RabbitMQ 怎么避免消息丢失：**

把消息持久化磁盘，保证服务器重启消息不丢失。每个集群中至少有一个物理磁盘，保证消息落入磁盘。

**要保证消息持久化成功的条件有哪些：**

声明队列必须设置持久化 durable 设置为 true.

消息推送投递模式必须设置持久化，deliveryMode 设置为 2（持久）。

消息已经到达持久化交换器。

消息已经到达持久化队列。

以上四个条件都满足才能保证消息持久化成功。

**RabbitMQ 持久化有什么缺点：**

持久化的缺地就是降低了服务器的吞吐量，因为使用的是磁盘而非内存存储，从而降低了吞吐量。可尽量使用 ssd 硬盘来缓解吞吐量的问题。

**RabbitMQ 有几种广播类型：**

direct（默认方式）：最基础最简单的模式，发送方把消息发送给订阅方，如果有多个订阅者，默认采取轮询的方式进行消息发送。

headers：与 direct 类似，只是性能很差，此类型几乎用不到。

fanout：分发模式，把消费分发给所有订阅者。

topic：匹配订阅模式，使用正则匹配到消息队列，能匹配到的都能接收到。

**RabbitMQ 怎么实现延迟消息队列：**

延迟队列的实现有两种方式：

通过消息过期后进入死信交换器，再由交换器转发到延迟消费队列，实现延迟功能。

使用 RabbitMQ-delayed-message-exchange 插件实现延迟功能。

**RabbitMQ 集群有什么用：**

集群主要有以下两个用途：

高可用：某个服务器出现问题，整个 RabbitMQ 还可以继续使用。

高容量：集群可以承载更多的消息量。

**RabbitMQ 节点的类型有哪些：**

磁盘节点：消息会存储到磁盘。

内存节点：消息都存储在内存中，重启服务器消息丢失，性能高于磁盘类型。

**RabbitMQ 集群搭建需要注意哪些问题：**

各节点之间使用“--link”连接，此属性不能忽略。

各节点使用的 erlang cookie 值必须相同，此值相当于“秘钥”的功能，用于各节点的认证。

整个集群中必须包含一个磁盘节点。

**RabbitMQ 每个节点是其他节点的完整拷贝吗？为什么？**

不是，原因有以下两个：

存储空间的考虑：如果每个节点都拥有所有队列的完全拷贝，这样新增节点不但没有新增存储空间，反而增加了更多的冗余数据；

性能的考虑：如果每条消息都需要完整拷贝到每一个集群节点，那新增节点并没有提升处理消息的能力，最多是保持和单节点相同的性能甚至是更糟。

**RabbitMQ 集群中唯一一个磁盘节点崩溃了会发生什么情况：**

如果唯一磁盘的磁盘节点崩溃了，不能进行以下操作：

不能创建队列

不能创建交换器

不能创建绑定

不能添加用户

不能更改权限

不能添加和删除集群节点

唯一磁盘节点崩溃了，集群是可以保持运行的，但你不能更改任何东西。

**RabbitMQ 对集群节点停止顺序有要求吗：**

RabbitMQ 对集群的停止的顺序是有要求的，应该先关闭内存节点，最后再关闭磁盘节点。如果顺序恰好相反的话，可能会造成消息的丢失。

**kafka 可以脱离 zookeeper 单独使用吗？为什么？**

kafka 不能脱离 zookeeper 单独使用，因为 kafka 使用 zookeeper 管理和协调 kafka 的节点服务器。

**kafka 有几种数据保留的策略？**

kafka 有两种数据保存策略：按照过期时间保留和按照存储的消息大小保留。

**kafka 同时设置了 7 天和 10G 清除数据，到第五天的时候消息达到了 10G，这个时候 kafka 将如何处理？**

这个时候 kafka 会执行数据清除工作，时间和大小不论那个满足条件，都会清空数据。

**什么情况会导致 kafka 运行变慢？**

cpu 性能瓶颈

磁盘读写瓶颈

网络瓶颈

**使用 kafka 集群需要注意什么？**

集群的数量不是越多越好，最好不要超过 7 个，因为节点越多，消息复制需要的时间就越长，整个群组的吞吐量就越低。

集群数量最好是单数，因为超过一半故障集群就不能用了，设置为单数容错率更高。

**zookeeper 是什么？**

zookeeper 是一个分布式的，开放源码的分布式应用程序协调服务，是 google chubby 的开源实现，是 hadoop 和 hbase 的重要组件。它是一个为分布式应用提供一致性服务的软件，提供的功能包括：配置维护、域名服务、分布式同步、组服务等。

**zookeeper 都有哪些功能？**

集群管理：监控节点存活状态、运行请求等。

主节点选举：主节点挂掉了之后可以从备用的节点开始新一轮选主，主节点选举说的就是这个选举的过程，使用 zookeeper 可以协助完成这个过程。

分布式锁：zookeeper 提供两种锁：独占锁、共享锁。独占锁即一次只能有一个线程使用资源，共享锁是读锁共享，读写互斥，即可以有多线线程同时读同一个资源，如果要使用写锁也只能有一个线程使用。zookeeper可以对分布式锁进行控制。

命名服务：在分布式系统中，通过使用命名服务，客户端应用能够根据指定名字来获取资源或服务的地址，提供者等信息。

**zookeeper 有几种部署模式？**

zookeeper 有三种部署模式：

单机部署：一台集群上运行；

集群部署：多台集群运行；

伪集群部署：一台集群启动多个 zookeeper 实例运行。

**zookeeper 怎么保证主从节点的状态同步？**

zookeeper 的核心是原子广播，这个机制保证了各个 server 之间的同步。实现这个机制的协议叫做 zab 协议。 zab 协议有两种模式，分别是恢复模式（选主）和广播模式（同步）。当服务启动或者在领导者崩溃后，zab 就进入了恢复模式，当领导者被选举出来，且大多数 server 完成了和 leader 的状态同步以后，恢复模式就结束了。状态同步保证了 leader 和 server 具有相同的系统状态。

**集群中为什么要有主节点？**

在分布式环境中，有些业务逻辑只需要集群中的某一台机器进行执行，其他的机器可以共享这个结果，这样可以大大减少重复计算，提高性能，所以就需要主节点。

**集群中有 3 台服务器，其中一个节点宕机，这个时候 zookeeper 还可以使用吗？**

可以继续使用，单数服务器只要没超过一半的服务器宕机就可以继续使用。

**说一下 zookeeper 的通知机制？**

客户端端会对某个 znode 建立一个 watcher 事件，当该 znode 发生变化时，这些客户端会收到 zookeeper 的通知，然后客户端可以根据 znode 变化来做出业务上的改变。