**分布式相关**

**分布式事务**

CAP定理

​ 分布式的不可能全部满足三个需求

​ 一致性（C：Consistency）

​ 可用性（A：Availability）

​ 分区容错性（P：Partition tolerance）

​ CA

​ 放弃分区容错性，加强一致性和可用性，其实就是传统的单机数据库的选择

​ CP

​ 放弃一致性（这里说的一致性是强一致性），追求分区容错性和可用性，这是很多分布式系统设计时的选择，例如很多NoSQL系统就是如此

​ AP

​ 放弃可用性，追求一致性和分区容错性，基本不会选择，网络问题会直接让整个系统不可用

​ 分布式事务可以使用多个方案

Base(Basically Available，Soft state,Eventually consistent)：一种 ACID的替代方案

​ Basically Available（基本可用）

​ 基本可用是指分布式系统在出现不可预知故障的时候，允许损失部分可用性----注意，这绝不等价于系统不可用 ，比如响应时间慢了1秒或者下订单失败进入失败页面，其实这就是一个强一致性

​ Soft state（软状态）

​ 软状态指允许系统中的数据存在中间状态，并认为该中间状态的存在不会影响系统的整体可用性，即允许系统在不同节点的数据副本之间进行数据同步的过程存在延时 ，也就是数据同步时间不一致，也称弱一致性

​ Eventually consistent（最终一致性）

​ 最终一致性强调的是所有的数据副本，在经过一段时间的同步之后，最终都能够达到一个一致的状态。因此，最终一致性的本质是需要系统保证最终数据能够达到一致，而不需要实时保证系统数据的强一致性。当这个程序执行完成之后所有的数据都一致，这个是最终一致性

**2PC**

​ 2PC是基于XA协议的两阶段提交协议（Two- Phase Commit Protocol）

​ 预提交阶段(Pre-Commit Phase)

​ 两阶段提交在应用程序向协调者发出一个提交命令时被启动。这时提交进入第一阶段，即预提交阶段 .协调者准备局部（即在本地）提交并在日志中写入"预提交"日志项，并包含有该事务的所有参与者的名字 ,如果全部都提交答复并且不否定则进入到第二个提交阶段，如果其中有一个参与者不能参与答复，或者否定了一个，那么就会取消事务，撤销他们的影响

​ 决策后阶段（Post-Decision Phase）

​ 如果所有的参与者都送回"已准备好提交"的消息，则该事务的提交进入第二阶段，即决策后提交阶段，协调者在日志中写入"提交"日志项，并立即写入硬盘 ，然后送回"已提交"的消息，释放该事务占用的资源。 如果所有都回复已提交那么就完成事务的提交，如果提交发生异常则回滚

​ 优点：

​  尽量保证了数据的强一致，适合对数据强一致要求很高的关键领域。

​ 缺点：

​ 实现复杂，牺牲了可用性，对性能影响较大，涉及多次节点间的网络通信，通信时间太长，不适合高并发高性能场景

**TCC**

​ TCC（Try Confirm Cancel） 其实就是采用的补偿机制，

​ Try

​ 这里是对业务系统做检测及资源预留

​ Confirm

​ 在 Confirm 阶段，执行远程操作，如果成功则直接成功，如果失败则执行Cancel

​ Cancel

​ 主要是在业务执行错误，需要回滚的状态下执行的业务取消，预留资源释放

​ 优点：实现简单，方便

​ 缺点：数据的一致性差，并且需要写非常多的补偿代码，在一些场景中，一些业务流程可能用TCC不太好定义及处理

**CAP**

**MQ 事务消息**

​ 有一些第三方的 MQ 是支持事务消息的，比如 RocketMQ，他们支持事务消息的方式也是类似于采用的二阶段提交

​ 第一阶段 Prepared 消息，会拿到消息的地址。

​ 第二阶段执行本地事务

​ 第三阶段通过第一阶段拿到的地址去访问消息，并修改状态。

​ ， 一次发送消息和一次确认消息。如果确认消息发送失败了 RocketMQ 会定期扫描消息集群中的事务消息，这时候发现了 Prepared 消息，它会向消息发送者确认，所以生产方需要实现一个 check 接口，RocketMQ 会根据发送端设置的策略来决定是回滚还是继续发送确认消息。

​ 优点： 实现了最终一致性，不需要依赖本地数据库事务。

​ 缺点： 实现难度大，主流MQ不支持，没有.NET客户端，RocketMQ事务消息部分代码也未开源。

**分布式锁如何设计**

​ 分布式锁的设计有三种

​ 数据库

​

​ Redis

​ 使用setnx并且设置超时时间，只有一个人能拿到锁，拿到了锁就直接进行数据库查询然后将数据添加到Redis当中，然后没有拿到锁的先睡眠一秒（防止还是没有数据，一秒钟可以将数据存入缓存），然后调用自己的方法回去，现在再次查询redis已经是有数据的了，这样就实现了分布式查询锁

​ Zookeeper

​

**分布式session如何设计**

​ 分布式session的实现方式有很多种

​ 基于数据库的session共享

​ 基于Redis的session缓存

​ 基于cookie的session共享

​ 这里我们推荐使用Redis，因为数据库性能并不高，而且我们也需要减少数据库压力，而cookie则容易被获取，从而造成数据的安全隐患，所以使用Redis既能优化速度，也能保证安全，还能方便清理

**dubbo的组件有哪些**

​ Remoting: 网络通信框架，实现了 sync-over-async 和request-response 消息机制.

​

​ RPC: 一个远程过程调用的抽象，支持负载均衡、容灾和集群功能

​

​ Registry: 服务目录框架用于服务的注册和服务事件发布和订阅

**dubbo是如何利用接口就可以通信的**

那我们就先来从dubbo的在通信中使用到的数据吧，

​ **Protocol：**注册协议，包括zookeeper、multicast、Redis、simple；

​ Address：\*\*注册地址，dubbo服务的IP+Port：

​ ①、当使用zk，address填入zk地址，集群地址使用","分隔；

​ ②、使用dubbo直连，address填写直连地址和服务端口；

​ **Protocol：**使用的dubbo协议，包括dubbo、rmi、hessian、webservice、memcached、redis，根据自己的协议类型选择对应的选项即可；

​ **Timeout：**请求超时时间，单位ms，根据dubbo具体配置填写；

​ Version：\*\*版本，dubbo不同版本之间差异较大，不同版本之间不能互相调用，这里指定dubbo版本，是为了方便识别和说明；

​ **Retries：**异常重试次数（类似这种分布式服务通信框架，大多都有重试机制，是为了保证事务成功率）；

​ **Cluster：**集群类型，包括failover、failfast、failsafe、failback、failking；

​ **Group：**组类型，如果有的话，根据配置填写即可；

​ **Connections：**连接数，同上，根据配置填写；

​ **Async：**服务处理类型，包括sync（同步）、async（异步），根据配置填写；

​ **Loadbalance：**负载均衡策略，包括random（随机）、roundrobin（轮询）、leastactive（最少活跃数）、consistenthash（一致性哈希）；

​ **Interface：**接口名（因为dubbo服务大多是开发根据规范自行命名的，因此这里需要填写完整的接口名+包名）；

​ **Method：**当前接口下的方法名，按照开发提供的API文档填写即可；

​ **Args：**接口报文，根据API文档填写，如上图所示，添加输入行，输入对应的参数类型和值即可（参数类型和值如何定义填写，请参考上面的链接）；

​ ①、paramType：参数支持任何类型，包装类直接使用java.lang下的包装类，小类型使用：int``、``float``、``shot``、``double``、``long``、``byte``、``boolean``、``char，自定义类使用类完全名称；

​ ②、paramValue：基础包装类和基础小类型直接使用值，例如：int为1，boolean为true等，自定义类与List或者Map等使用json格式数据；

​ 这里我们可以看到他记录了非常多的东西，甚至ip以及协议，接口名，方法名还有参数，那么他回去进行匹配zookeeper的节点，如果匹配了就通过ip和接口还有方法名进行调用，他的调用的过程是不经过zookeeper的，因为我们拿到了ip和接口就可以直接进行rpc的远程调用，甚至与如果zookeeper挂掉了，其实他们地也是有缓存的还能继续使用，从这点可以看出并没有经过zookeeper，rpc的远程调用使用的原理是socket，那么他在通信是已经拿到了ip那么就可以直接调用他的服务了，这就是dubbo利用通信的原理

**Zookeeper在选举的过程中，还能对外提供服务么？**

​ 这个就需要结合情况看了，如果是单纯的使用zookeeper的话那肯定是对外提供不了服务了，但是如果是使用的dubbo的话那就是还能使用的，因为dubbo是将服务的注册信息都写入到了zookeeper的节点中，如果他开始就通过zookeeper调用过的话本地是有缓存的，当zookeeper在选举的过程中，如果他还是调用访问过的服务那就是没问题的，除非缓存过期了，

**Web相关**

**Cookie和Session的区别**

Cookie以文本文件格式存储在浏览器中，而session存储在服务端它存储了限制数据量。它只允许4kb它没有在cookie中保存多个变量。

cookie的存储限制了数据量，只允许4KB，而session是无限量的

我们可以轻松访问cookie值但是我们无法轻松访问会话值，因此它更安全

设置cookie时间可以使cookie过期。但是使用session-destory（），我们将会销毁会话。

**Serlvet的生命周期**

初始化 ---》 运行 ---》 销毁

初始化 init()

​ init 方法被设计成只调用一次。它在第一次创建 Servlet 时被调用，在后续每次用户请求时不再调用，当用户调用一个 Servlet 时，但是您也可以指定 Servlet 在服务器第一次启动时被加载 ，就会创建一个 Servlet 实例，每一个用户请求都会产生一个新的线程，适当的时候移交给 doGet 或 doPost 方法

运行 service()

​ service() 方法是执行实际任务的主要方法。Servlet 容器（即 Web 服务器）调用 service() 方法来处理来自客户端（浏览器）的请求，并把格式化的响应写回给客户端。 每次服务器接收到一个 Servlet 请求时，服务器会产生一个新的线程并调用服务。service() 方法检查 HTTP 请求类型（GET、POST、PUT、DELETE 等），并在适当的时候调用 doGet、doPost、doPut，doDelete 等方法。

销毁 destroy()

​

​ destroy() 方法只会被调用一次，在 Servlet 生命周期结束时被调用。destroy() 方法可以让您的 Servlet 关闭数据库连接、停止后台线程、把 Cookie 列表或点击计数器写入到磁盘，并执行其他类似的清理活动。在调用 destroy() 方法之后，servlet 对象被标记为垃圾回收。

**Jsp和Servlet的区别**

​ jsp经编译后就变成了Servlet，jsp更擅长表现于页面显示,servlet更擅长于逻辑控制，Jsp是Servlet的一种简化，使用Jsp只需要完成程序员需要输出到客户端的内容，Jsp中的Java脚本如何镶嵌到一个类中，由Jsp容器完成，Servlet更多的是类似于一个Controller，用来做控制

​ 区别：

​ Servlet在Java代码中通过HttpServletResponse对象动态输出HTML内容

​ JSP在静态HTML内容中嵌入Java代码，Java代码被动态执行后生成HTML内容

**Tomcat是什么**

​ Tomcat是一个应用端的Web服务器，属于轻量级应用服务器，tomcat除了实现了http协议，用来接收和响应Http请求

**Tomcat和Servlet的区别**

​ ，Tomcat实际上也是一个Servlet容器，但是这个端是在Web服务层的，而Java还需要经过动态的响应以及处理需要和Web服务端进行交互，那么这个交互的容器就是Servlet，而Servlet就会进行一系列的处理响应给Tomcat

**Request和Response的区别是什么**

​ Request对象是服务器对浏览器所发送过来的请求的封装，而Response是是服务器对浏览器的响应的对象，里面所封装了请求头和响应头，封装了请求的请求信息，以及响应的响应信息

**四大域和九大内置对象**

​ Request域

​ request请求对象

​ Page域

​ response响应对象

​ pageContext页面上下文对象

​ out输出对象

​ config配置对象

​ page页面对象

​ exception例外对象

​ Application域

​ application应用程序对象

​ Session域

​ session会话对象

**缓存相关**

**redis和memcached的区别**

​ 相对于 Memcached，来说如果他断电了数

​ 据也就没有了，不能像Redis一样支持数据的持久化，在这一点上Redis做的比他好，并且它包含了

​ Memcached的几乎所有的功能并且比他更加强大，作为Java来说Redis是一个不错的缓存数据库，

​ 这只是Redis的一部分功能，还有更多的强大的功能，并且Memcached只支持json格式的字符串，

​ 但是Redis支持很多种数据类型，所以在现在的市场上Memcached已经逐渐被Redis取代了

**redis支持哪些数据结构**

​ 五种数据类型：

​ 字符串（String）

​ 哈希（hash）

​ 字符串列表（list）

​ 字符串集合（set）

​ 有序字符串集合（zset sorted set）

​ 但是在Redis5.0中发布了新的数据类型

​ Stream data type

**redis是单线程的么，所有的工作都是单线程么**

​ Redis是单线程的，redis实际上是采用了线程封闭的观念，把任务封闭在一个线程，自然避免了线程安全问题，不过对于需要依赖多个redis操作的复合操作来说，依然需要锁，而且有可能是分布式锁。 所有的工作并不是单线程的，而是使用I/O的多路复用，

​ Redis 是跑在单线程中的，所有的操作都是按照顺序线性执行的，但是由于读写操作等待用户输入或输出都是阻塞的，所以 I/O 操作在一般情况下往往不能直接返回，这会导致某一文件的 I/O 阻塞导致整个进程无法对其它客户提供服务，而 I/O 多路复用就是为了解决这个问题而出现的

​ 那么什么是多路I/O的复用呢？

​ 多路指的是多个TCP连接，复用指一个或者多个线程进行处理，在Redis中是单线程的，简单的理解就是多个TCP连接进行数据写入的请求，通过一个单线程进行多路的IO写入，这个就是多路IO复用，但是I/O的多路复用也是有很多种模型的，常见的有、 select 、 poll 、epoll

​ 而在Redis当中使用的epoll

**redis如何存储一个String的**

​ 通过他的Key Value的键值对进行存储，

**一个key值如何在redis集群中找到存储在哪里**

​ 一个 Redis 集群包含 16384 个插槽（hash slot）， 数据库中的每个键都属于这 16384 个插槽的其中一个， 集群使用公式 CRC16(key)% 16384 来计算键 key 属于哪个槽， 其中 CRC16(key) 语句用于计算键 key 的 CRC16 校验和 。• 集群中的每个节点负责处理一部分插槽。 举个例子， 如果一个集群可以有主节点， 其中：节点 A 负责处理 0 号至 5500 号插槽。节点 B 负责处理 5501 号至 11000 号插槽。节点 C 负责处理 11001 号至 16383 号插槽。

​ 那么redis如果有3个节点，就会根据他的key进行计算key的CRC16的值然后来对16384 进行取模的操作

​ 也就是（Key 计算后的CRC16 % 16384），然后再计算取模后的值在哪一个节点中，例如取模后是3000，那么他就会找到3000的这个槽，3000的这个槽是在A节点（1-5500）（假设为3个节点），那么他就会在第一个节点进行存储，如果查找的话也会去第一个节点进行查询

**redis持久化策略**

**RDB**

​ RDB的存储方式：在指定的时间间隔内将内存中的数据集快照写入磁盘，也 就是行话讲的Snapshot快照，它恢复时是将快照文件直接 读到内存里

​ Redis会单独创建（fork）一个子进程来进行持久化，会先将数据写入 到一个临时文件中，待持久化过程都结束了，再用这个临时文件替换 上次持久化好的文件。整个过程中，主进程是不进行任何IO操作的， 这就确保了极高的性能如果需要进行大规模数据的恢复，且对于数据 恢复的完整性不是非常敏感，那RDB方式要比AOF方式更加的高效。 RDB的缺点是最后一次持久化后的数据可能丢失

​ rdb的保存的文件： 在redis.conf中配置文件名称，默认为dump.rdb

​

​ 下面就是他的持久化的文件存储的路径

​ RDB持久化的保存策略

​ 这是他的保存策略，如果60秒内发生了10000次数据操作则进行一次存储，

​ 如果300秒发生了10次则会存入一次。如果900秒内发生了一次操作那么900秒后

​ 就会备份一次，

​

​ stop-writes-on-bgsave-error yes 当Redis无法写入磁盘的话，直接关掉Redis的写操作

​ rdbcompression yes 进行rdb保存时，将文件压缩

​ rdbchecksum yes 在存储快照后，还可以让Redis使用CRC64算法来进行数 据校验，但是这样做会增加大约10%的性能消耗，如果希 望获取到最大的性能提升，可以关闭此功能

​ rdb的备份

​ 先通过config get dir 查询rdb文件的目录 将\*.rdb的文件拷贝到别的地方

​ rdb的恢复

​ 先把备份的文件拷贝到工作目录下 关闭Redis 启动Redis, 备份数据会直接加载

​ 注：清先看清楚配置文件的RDB的文件名和路径

​ 优点：

​ 节省磁盘空间 恢复速度快

​ rdb的缺点

​ 虽然Redis在fork时使用了写时拷贝技术,

​ 但是如果数据庞大时还是比较消耗性能

​ 在备份周期在一定间隔时间做一次备份，所以如果 Redis意外down掉的话，就会丢失最后一次快照后的所有修改

**AOF**

​ AOF默认不开启，需要手动在配置文件中配置

​ 可以在redis.conf中配置文件名称，默认为 appendonly.aof

​

​ 这分别是是否开启。默认是关闭的而RDB默认是开启的，他的路径和RDB是一样的

​ 下面这个就是文件名了

​

​ 那么如果RDB好AOF同时启动的话他会执行哪个呢？

​ AOF的备份机制和性能虽然和RDB不同, 但是备份和 恢复的操作同RDB一样，都是拷贝备份文件，需要 恢复时再拷贝到Redis工作目录下，启动系统即加载

​ **AOF和RDB同时开启，系统默认取AOF的数据**

​ 持久化AOF

​ AOF文件的保存路径，同RDB的路径一致

​ 如遇到AOF文件损坏，可通过 redis-check-aof --fix appendonly.aof 进行恢复

​ AOF优点：

​ 恢复数据全，不会丢失数据

​ 缺点：

​ 占用磁盘空间较大，并且恢复没有RDB快

* 如果只配置AOF，重启时加载AOF文件恢复数据；
* 如果同时配置了RBD和AOF，启动是只加载AOF文件恢复数据;
* 如果只配置RBD，启动时将加载dump文件恢复数据。

**你们用了redis，redis的底层数据结构了解多少？**

​ 简单动态字符串

​ 链表

​ 字典

​ 跳跃表

​ 整数集合

​ 压缩列表

**知道动态字符串sds的优缺点么？ 注:sds为redis底层数据结构之一**

​ 核心：

​ 更加高效的执行长度计算：

​ 在C中获取长度是需要进行遍历的，但是在sds中直接获取size属性，时间复杂度为O(1)，C的原生字符串获取长度为O(n)

​ 更加高效的执行追加操作：

​ 他在拓展的时候会比实际的多，例如追加10但是他扩容了15个，这样的话他下面5次追加就不会进行扩容了，而且他使用了惰性空间释放，也就是例如多出了10个空间他会把这10个记录下来，留着下次追加使用，而不是直接将10个空间进行释放，但是这样会让我们的内存空间变大，这个也可以手动进行释放（在SDS）的Api中有记录

​ 二进制安全：

​ 因为C字符串以空字符作为字符串结束的标识，而对于一些二进制文件（如图片等），内容可能包括空字符串，因此C字符串无法正确存取；而所有 SDS 的API 都是以处理二进制的方式来处理 buf 里面的元素，并且 SDS 不是以空字符串来判断是否结束，而是以 len 属性表示的长度来判断字符串是否结束。

sds 会为追加操作进行优化：加快追加操作的速度，并降低内存分配的次数，代价是多占用了一些内存，而且这些内存不会被主动释放。

**redis的单线程特性有什么优缺点？**

优点：

1. 代码更清晰，处理逻辑更简单
2. 不用去考虑各种锁的问题，不存在加锁释放锁操作，没有因为可能出现死锁而导致的性能消耗
3. 不存在多进程或者多线程导致的切换而消耗CPU

缺点：

1. 无法发挥多核CPU性能，不过可以通过在单机开多个Redis实例来完善；

**用过 Redis 的哪些数据结构, 分别用在什么场景?**

**String**

​ String类型他并不是只能存储String，也能用来存储Long，等等，他可以用来存储一些结构简单的数据，并且效率非常高，像日常的一些单个的值往里面存，例如用户的token，还有用户的信息

**Hash**

​ Hash是一个字典类型，这个时候我们可以用它来存储一些需要频繁修改的对象，例如用户的积分（但一般也是在数据库中存储，Redis存储一般用来提高效率），那么我们需要对用户的积分进行改动，而且这种改动还比较频繁，那么我们可以使用Hash来进行存储这个用户对象，因为如果使用String的话他会全部的序列化出来再存储进去，这样造成的大量IO比较消耗性能

**List**

​ List是一个列表，他其实是一个链表，但是更多的使用时是将他当做一个消息队列来使用，因为他可以push和pop这和Java中的Queue队列一样，我们可以使用它来做一个消息队列

**Set**

​ Set是一个集合，他和Java中的Set也是一样的无序的不唯一的值，如果使用的话场景还是比较多的，例如向的好友列表，还有关注的用户，和粉丝，以及还能用来统计访问用户，比如统计某一天的用户访问量，和访问过的用户

**ZSet**

​ ZSet是一个有序的集合，他可以用来存储集合并且做一个排序，那么我们可以使用他来进行一个点赞的操作，应为他去重并且有序，但实际上来说ZSet的应用场景其实并不多

​ 这几个数据类型更多的是结合实际的业务需求来进行使用的

**你们怎么解决缓存击穿问题的？**

​ 使用分布式锁，设置ex以及过期时间，然后进行数据库查询并重新添加到Redis中，没有获取到锁的对象Seelp睡眠一秒然后重新调用方法，这样就解决了缓存击穿的问题