美团优选

美团优选2面

1.项目介绍

2.mq 使用场景

3.分库分表，原因，怎么分

4.分布式锁：redisson,zookeeper

5.redis lua ， 管道

6.跳表

7.项目中大的技术挑战

8.Springcloud源码 ， 组件.

9.你觉得比较好的源码，AQS，中间件，feign的源码实现过程

10. 熔断器的原理

11.分布式事务有哪几种方式，2pc,3pc,tcc,mq及怎么实现，优缺点

12.2pc TCC 区别

13.二分法算法

14. 程序：问打印出来的数值是多少?

15.堆应用场景

16.redis 消息队列和mq区别

17.redis分布式锁，有事务的还是没有事务的分布式锁

18.redis 分布式锁和zk 的分布式锁有啥区别

19.Redis中有2千万数据，怎么找出最热点的两万数据，

20.redis mysql数据一致性解决方案

21.spring 的循环依赖怎么解决

22.spring中单例bean 中线程安全有了解吗

23.spring事务，在配置上都没问题，在什么场景下不生效

24.rocketMQ 不使用ZK作为注册中心的原因了解过吗?

25.jms 和amqp 的区别?

26.线程池饱和策略

27.触发fullGC的场景

28.算法： 二叉树，按照前序，中序，后序，打印出来

美团优选2面

1.项目介绍

2.mq 使用场景

异步，削峰、解耦

3.分库分表，原因，怎么分

肯定是为了解决数据量大、承载更高并发量，mysql读写性能更不上。

水平拆分，垂直拆分，32*32=1024

4.分布式锁：redisson,zookeeper

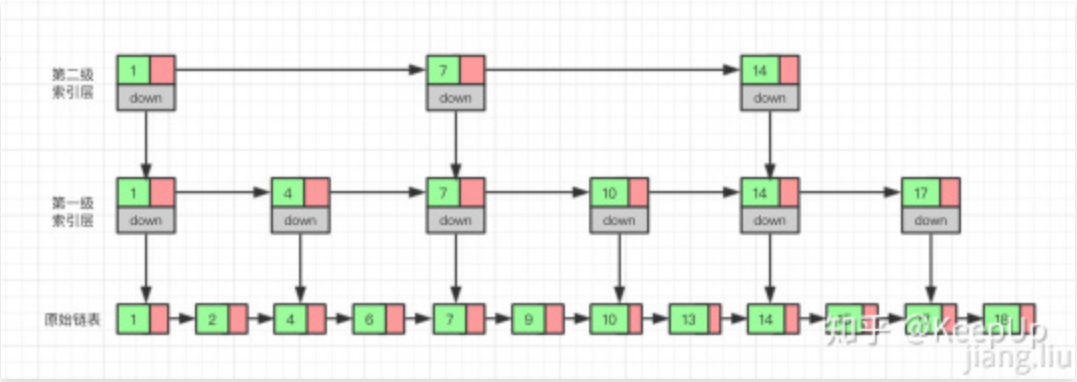
Rlock = redission.getLock("myLock");

5.redis lua ， 管道

> Redis中为什么引入Lua脚本？ c语言编写的脚本
>
> - 减少网络开销。可以将多个请求通过脚本的形式一次发送，减少网络时延。
> - 原子操作。Redis会将整个脚本作为一个整体执行，中间不会被其他请求插入。因此在脚本运行过程中无需担心会出现竞态条件，无需使用事务。
> - 复用。客户端发送的脚本会永久存在redis中，这样其他客户端可以复用这一脚本，而不需要使用代码完成相同的逻辑。
>
> Redis管道
>
> Redis在很早就支持了管道技术。也就是说客户端可以一次发送多条命令，不用逐条等待命令的返回值，而是到最后一起读取返回结果，这样只需要一次网络开销，速度就会得到明显的提升。
>
> 例如POP3协议就是支持管道技术，提升下载邮件的速度。

Redis2.6版本以后，脚本在大部分场景中的表现要优于管道。

6.跳表



这种通过对链表加多级索引的机构，就是跳表。实质就是一种可以进行二分查找的有序链表。

跳表时间复杂度：O(log(n))。

跳表的效率比链表高了，但是跳表需要额外存储多级索引，所以需要的更多的内存空间。（用空间换时间）

为什么Redis选择使用跳表而不是红黑树来实现有序集合set呢?

在跳表中，要查找区间的元素，我们只要定位到两个区间端点在最低层级的位置，然后按顺序遍历元素就可以了，非常高效。而红黑树只能定位到端点后，再从首位置开始每次都要查找后继节点，相对来说是比较耗时的

7.项目中大的技术挑战

8.Springcloud源码 ， 组件.

9.你觉得比较好的源码，AQS，中间件，feign的源码实现过程

10. 熔断器的原理

Hystrix在底层使用了Spring提供的切面技术。

11.分布式事务有哪几种方式，2pc,3pc,tcc,mq及怎么实现，优缺点

12.2pc TCC 区别

- 2pc 和3pc 是依靠mysql xa 实现的，是db 层实现（就是直连mysql），
- tcc 业务层实现符合soa理念，解耦。也更稳定.

TCC属于业务上的分段提交，Try，confirm，cancel都是对应的一段业务逻辑的操作，先预留资源，预留成功后进行确认，不成功就取消，例如转账先冻结资金，进行一系列的余额各方面的检查，发现符合条件就将账户资金状态改为冻结，确认阶段修改状态为扣除，取消的话就把冻结的资金加回原账户，其对应的数据库的操作每段都是一个完整的事物；2PC是属于数据库层面的，先进行prepare，然后逐个进行commit或者rollback，不和具体业务逻辑挂钩，TCC的应用范围更广，不一定是关系型数据库，也可能操作的KV数据库，文档数据库，粒度也可以随着具体业务灵活调整，性能更好。

13.二分法算法

14. 程序：问打印出来的数值是多少?

Java 复制代码

```java
synchronize(xx){
  wait();
  System.out.println("a");
}

synchronize(xx){
  notifyall();
  System.out.println("b");
}
```

15.堆应用场景

数据结构中的堆是一种特殊的二叉树

堆必须符合以下两个条件：

1. 是一棵完全二叉树。
2. 任意一个节点的值都大于（或小于）左右子节点的值。

应用场景：

1. 优先级排序
2. top K

16.redis 消息队列和mq区别

- redis-pub/sub模式多用于实时性较高的消息推送，并不保证可靠。断电内存就清空，而使用redis-list作为消息推送虽然有持久化，但是又太弱智。专业消息队列都有持久化机制。
- redis 发布订阅除了表示不同的 topic 外，并不支持分组，比如kafka中发布一个东西，多个订阅者可以分组，同一个组里只有一个订阅者会收到该消息，这样可以用作负载均衡。
- redis持久化是整个内存一起持久化，mq是可以选择性的持久化粒度更细。
- mq实时性低，redis实时性高

17.redis分布式锁，有事务的还是没有事务的分布式锁

没有事务

18.redis 分布式锁和zk 的分布式锁有啥区别

19.Redis中有2千万数据，怎么找出最热点的两万数据，

不能用zset ,因为最大key 为1M，存不下，应该是用redis 的lru算法，设置redis内存大小，设置内存淘汰策略为LRU。

20.redis mysql数据一致性解决方案

- 延时双删机制
- 延时双删+重试
- MySQL binlog增量发布订阅消费+消息队列+增量数据更新到redis
  （1）更新数据库数据
  （2）数据库会将操作信息写入binlog日志当中
  （3）订阅程序提取出所需要的数据以及key
  （4）另起一段非业务代码，获得该信息
  （5）尝试删除缓存操作，发现删除失败
  （6）将这些信息发送至消息队列
  （7）重新从消息队列中获得该数据，重试操作。

上述的订阅binlog程序在mysql中有现成的中间件叫canal，可以完成订阅binlog日志的功能。如果对一致性要求不是很高，直接在程序中另起一个线程，每隔一段时间去重试即可代替mq。

21.spring 的循环依赖怎么解决

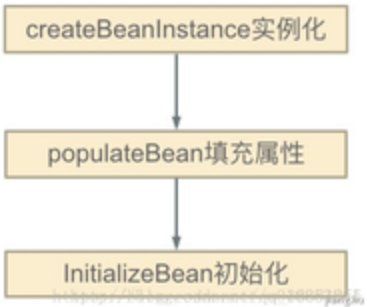
三级缓存

循环依赖场景：

①：构造器的循环依赖。【这个Spring解决不了】

②：setter循环依赖，field属性的循环依赖

Spring的单例对象的初始化主要分为三步：



①：createBeanInstance：实例化，其实也就是 调用对象的构造方法实例化对象

②：populateBean：填充属性，这一步主要是多bean的依赖属性进行填充

③：initializeBean：调用spring xml中的init() 方法。

从上面讲述的单例bean初始化步骤我们可以知道：循环依赖主要发生在第一、第二步。也就是构造器循环依赖和field循环依赖。

调整配置文件，将构造函数注入方式改为 属性注入方式 即可。

A的某个field或者setter依赖了B的实例对象，同时B的某个field或者setter依赖了A的实例对象，这种循环依赖的情况。A首先完成了初始化的第一步，并且将自己提前曝光到三级缓存 singletonFactories 中，此时进行初始化的第二步，发现自己依赖对象B，此时就尝试去get(B)，发现B还没有被 create，所以走 create 流程，B 在初始化第一步的时候发现自己又依赖了对象 A，于是尝试 get(A)，尝试一级缓存 singletonObjects(肯定没有，因为A还没初始化完全)，尝试二级缓存 earlySingletonObjects（也没有），尝试三级缓存 singletonFactories，由于A通过ObjectFactory将自己提前曝光了，所以B能够通过 ObjectFactory.getObject拿到A对象(虽然A还没有初始化完全，但是总比没有好呀)，B 拿到 A 对象后顺利完成了初始化阶段 1、2、3，完全初始化之后将自己放入到一级缓存 singletonObjects 中。此时返回 A 中，A 此时能拿到 B 的对象顺利完成自己的初始化阶段 2、3，最终 A 也完成了初始化，进去了一级缓存

singletonObjects 中，而且更加幸运的是，由于 B 拿到了 A 的对象引用，所以 B 现在hold住 A 对象完成了初始化。

知道了这个原理时候，肯定就知道为啥Spring不能解决“A的构造方法中依赖了B的实例对象，同时B的构造方法中依赖了A的实例对象”这类问题了！因为加入 singletonFactories 三级缓存的前提是执行了构造器，所以构造器的循环依赖没法解决。

22.spring中单例bean 中线程安全有了解吗

线程不安全，可以通过设置bean的作用域为prototype，请求bean相当于重新new bean。

23.spring事务，在配置上都没问题，在什么场景下不生效

- **数据库引擎不支持事务**，确认必须是InnoDB，MyISAM不支持事务
- **没有被 Spring 管理**，比如ServiceImpl代码中如果不加@service主键，就不会将bean注入spring容器，当然不会被spring管理。
- **调用的方法必须是public**，`@Transactional` 只能用于 public 的方法上。否则事务不起作用。 如果要用在非 public 方法上，可以开启 `AspectJ` 代理模式。
- **Spring的事务传播策略在内部方法调用**时将不起作用。因为它们发生了自身调用，就是调该类自己的方法，而没有经过 Spring 的代理类，默认只有在外部调用事务才会生效。
- **数据源没有配置事务管理器**。
- **事务传播机制设置为不支持事务** `NOT_SUPPORTED`。
- **事务中发生异常被吃了**，就是被代码主动 `try catch` 了。
- **事务中异常抛出的类型不**对，与事务默认回滚类型对不上。

24.rocketMQ 不使用ZK作为注册中心的原因了解过吗?

根据CAP理论，同时最多只能满足两个点，而zookeeper满足的是CP，也就是说zookeeper并**不能保证**服务的**可用性**，zookeeper在进行选举的时候，整个选举的时间太长，期间整个集群都处于不可用的状态，而这对于一个注册中心来说肯定是不能接受的，作为服务发现来说就应该是为可用性而设计。

25.jms 和amqp 的区别?

通信平台的区别

- JMS：只允许基于JAVA实现的消息平台的之间进行通信

- AMQP: 允许多种技术同时进行协议通信。

通信机制的区别

- JMS: 消息生产者和消息消费者必须知道对方的Queue。
- AMQP: 消息生产者和消息消费者无须知道对方的Queue,消息生产者将Exchange通过Route key和任意Queue绑定。消息消费者通过Route key从任意Queue中获取Exchange。

消息传输机制的区别

- JMS: JMS支持PTP和pub/sub机制,PTP只可以点对点通信, pub/sub在一端发出请求后所有其他端收到消息。
- AMQP:

  1 所有RouteKey相同的Queue接受到数据

  2 所有相同的Exchange的Queue接受到数据

  3 所有wilecard的Exchange的Queue接受到数据

  4 可以让webservice等接受到数据

26.线程池饱和策略

4个拒绝策略

- 抛异常
- 将新来的任务回退给调用者（主线程）去执行。
- 抛弃最早进入队列中的任务
- 抛弃当前任务

27.触发fullGC的场景

- 显式调用System.gc()方法，可能会触发Full GC。
- 老年代空间不足。
- 1.7的话，永久代空间不足会导致fullGC。
- 1.8的话，当Metaspace空间不足进行回收时，需同时满足如下三个条件的类才会被卸载：该类所有的实例都已经被回收、加载该类的ClassLoader已经被回收、该类对应的java.lang.Class对象没有任何地方被引用，进而触发Full GC。
- Minor GC晋升到老年代的平均大小大于老年代的剩余空间。
- Minor GC的时候，新生代Survivor空间不足，需要放入老年代，而老年代空间也不足，则触发Full GC。
- 堆中分配很大的对象，比如大数组，

28.算法： 二叉树，按照前序，中序，后序，打印出来

LeetCode