1.freemark

2.消息推从:https://cloud.tencent.com/developer/article/2094125

3.定时发布，延迟队列:DelayQueue

4.word生成

5.硬件和软件建立通讯

6.dlms协议

7.tcp/ip

8.elastricsearch

9.分布式锁 ：redis 分布式锁：setnx 设置key，如果能获取到这个key怎说明这个锁已经被占用，如果没有查询到这个key则这个锁没有获取。并设置key的失效时间

10.高并发多线程

11.xx-job:https://blog.csdn.net/qq\_57581439/article/details/128319069

12.2000kw的数据如何优化查询:

分区：

添加分区：

ALTER TABLE sales

ADD PARTITION sales\_2024\_q1 VALUES LESS THAN (TO\_DATE('2024-04-01', 'YYYY-MM-DD'));

删除分区：

ALTER TABLE sales

DROP PARTITION sales\_2023\_q1;

查询分区表的数据：

SELECT \* FROM sales PARTITION (sales\_2023\_q1);

13.i/o:

14.jvm:

a.生成dump文件，使用jps查看进程

b.导入到jvisulavm 中

15.java线程池:

16.webservice:

17.spring事务，什么时候事务会失效：

REQUIRED:required

A,B连个事务，互相管理，A异常那么，AB一起回滚，B异常 AB一起回滚

REQUIRED\_new:required\_new

A,B两个事务互不相关，A事务发生异常A回滚 B不会滚 B发生异常，如果A抓取异常，则A不会滚，B回滚

如果A不抓取，则AB都回滚

NESTED：nested 当A 发生异常 AB一起回滚 当B发生异常 A如果抓取异常则B回滚 A不回滚

事务失效：

1.数据库不支持事务

2.修饰的方法不是public

3.调用事务的方法没有用@Transcation

4.@transcation注解的类没有被bean管理

18.kafka集群如何顺序提交

a.相同的业务的消息放在同一个分区

b.一个分区用一个消费者去消费

c.acks=all

19.如何防止业务重复提交

kafka 的幂等性：enable.idempotence 启用幂等性开关，kafka会给每个消息分一个Producer ID 这样可以保证数据不会重复数据。

20.接口幂等性

1.在请求的时候设计加上一个唯一标识，这样多次请求的时候的，会根据找个唯一标识进行判断，然后筛选。

幂等性的特点：a.多次调用结果是一直的b.多次请求后状态不变。

接口幂等性设计

1.前端重复提交处理

2.进入表单前，服务端先把token存入redis，然后再删除redis，如果token被删除成功，则处理业务，如果删除不成功，说明重复提交了

3.数据库中给需要操作的数据加行锁，或者设置一个字段做乐观锁

4.redis加分布式锁

21.nginx 的config 配置：1.upstream 里面配置负载均衡的策略:1.将连接分配到最少的连接的服务上去2.根据权重分配请求3.将同一个请求分配到相同的服务器上去

severs:配置端口

22.kafka生产者如何制定消息发送到对应的分区：根据自定义key 或者自己制定分区

23.kafka的消费者如何指定分区进行消费：可以根据List<TopicPartition> partitions = Arrays.asList(new TopicPartition("your\_topic", 0), // 分区 0

new TopicPartition("your\_topic", 1)); // 分区 1

consumer.assign(partitions);

手动制定分区进行消费。

24.http和https协议

25.hashMap和TreeMap：hashMap 底层数组+链表+红黑树，key是不可重复的可以为一次null，value可以为null

TreeMap:可以根据自定义的排序接口进行自动排序。

26.nacos的分配请求的原理

27.redis的事务

a.MULTI:开启事务

b.执行事务操作

c.EXEC提交事务

d.DISCARD 取消事务

E.WATCH 监听KEY,如果KEY有变化则DISCARD上面的事务

28.线程的异步通信如何查看返回状态：future和callable

线程池：Future<Integer> future = exectutor.submit(callcable);

//获取线程执行的结果

Future.get()

29.kafka确保生产者不丢失数据：

a.acks=all

b.retries=MAX 生产者生产失败，不断重试。

30.确保消息队列不丢数据：

a.replication.factor参数，这个值必须大于1，即要求每个partition必须有至少2个副本

b.min.insync.replicas参数，这个值必须大于1，这个是要求一个leader至少感知到有至少一个follower还跟自己保持联系

31.消费者丢失数据:

将自动提交改成手动提交

32.jvm垃圾回收器的选择：

a.单线程垃圾收集器。

b.年轻代：复制算法处理

c.老年代：标记整理算法处理

d.并发类的是多线程垃圾收集器

33.堆里面的对象是线程共享的

34.方法区里面存储的类的信息和一个静态变量和常量

35.虚拟机栈里面的数据是线程私有的

36.抽象方法和普通方法

a.抽象类不能实例化

b.普通类继承抽象类必须实现抽象类的所有方法，普通类的继承不需要这样。

c.

37.java基础数据类型有几个字节

38.mysql有几个索引

1.主键索引，一般物理顺序和索引顺序一样，不能有null

2.唯一索引，不能重复,可以是null

3.hash索引，适合精准匹配，不适合范围查询

4.普通索引允许保存null

5.b-tree索引：适合做条件范围查询也适合左精确查询

b-tree:索引会把数据分为一页一页的，如果索引是主键（聚簇索引），则查询条件的数据值会跟索引筛选出的每一页的最后一个数值匹配，然后确定在那一页，然后在这一页查询。

非聚簇索引：则根据上面那样去寻找，只是第一步找到的是这行数据的主键，然后根据主键再去寻找对应的数据。

6.FULLTEXT：适合做txt 和varchar内容很长的字段索引，可以进行全文检索

7.复合索引：也是B\_tree

8. 聚簇索引：每个表只有一个，按照物理存储顺序存储

索引失效：

1.索引的数据类型不一致

2.复合索引的没有左匹配原则

3.索引的条件加了函数

4.在复合索引的时候BETWEEN 的是复合索引第一列然后第二列是复合索引的第二个数据的时候，当 组合索引第一列查询的数据中第二列的属性的数据不是按顺序来的，那么第二列的属性是会失效的。（>和<也是这样的）

5.索引like使用了左边使用了%

6.索引使用了or

7.全表扫描效率和用索引的效率差不多

39.数据库的事务

40.springAOP

41.spring的几种注入方式

42.介绍一下泛型

43.实现线程的几种方法并start() 和run() 的区别

44.面向对象为什么封装

45.java为什么不能多继承

46.线程池有多少种：

a. FixedThreadPool:固定了线程的数，超出线程池数量的就等待就好了。

b. CachedThreadPool：根据需要创建线程

c. SingleThreadExecutor：就一个线程

d. ScheduledThreadPool：定期执行的线程池

e. WorkStealingPool：会常见多个队列来运行线程，他使用多个队列来减少竞争

f. ForkJoinPool：将原来的任务分开两半，然后一起执行然后再将结果合并。

47.延迟队列的实现方法：

a.重新DelayQueue中的delay方法，这个方法确定了这个队列到期时间执行还剩多久

b.在自己的代码中用while（queue.isNotEmpty()）来不断的循环来获取队列中的数据。

等delay方法等于0的时候获取队列。

c.插入队列中的数据会根据时间的早晚排序。

48.数据库的锁：

a.

行锁：

lock in share mode

共享锁：允许事务进行读操作，其他操作不允许

For update

排他锁：只允许一个事务进行读写操作，其他事务不允许操作。

49.数据库的事务

1.原子性：事务不可以分割，要么全执行，要么全不执行

2.隔离性：事务之间互相不干扰

3.持久行：读完要保存

4.一致性:读完前后的结果是一致的

未提交读:会脏读：读到未提交的数据

提交读：避免脏读，但是可能一个事务中前后读取的数据不一致，就是在操作的前后，别的事务的数据也修改了数据

可重复度：可能出现幻读

串行执行：通过锁表完成了幻读，通过行锁，解决了不可重复读。

不可重复是在同一个事务中，读取同一条数据两次读取不同的结果

幻读是同一个事务中，查询的结果集可能前后不一致

50.多线程：

读锁：ReadWriteLock

写锁： ReadWriteLock

乐观锁：cas

悲观锁：synchronized，ReentrantLock

CountDownLatch: 设置初始计数，然后countDown（），直接到0，被调用await()的线程开始被唤醒调用。

Sleep:释放cpu

Yield:释放cpu

Join:调用者等待被调用的线程先执行

Wait:等待其他线程执行完，不让出cpu

Notify:唤醒等待的线程执行

51.微服务：

Gateway(路由，限流), **Hystrix(限流)，feigin,nacos,springboot**

**微服务的限流:设置的最大可以连接线程数，最大处理时间**

**微服务的熔断:@ hystrixCommond 后面又fallbackmethod=method就是熔断后调用的方法（服务的错误率高）**

**微服务的降级：fegin接口调用的时候fallbackFactory = method 实现FallBackFactory的时候实现create方法（服务的负载高）**

52.hashtable:线程安全的，跟hashmap不一样的是它的key和value不能为null

53.redis 的数据结果：list string hash,set zset

54.redis 缓存击穿：1.不要处置key值同一时间都失效

2.对查询的key值进行判断校验

3.对数据不存在的key进行key的值为null的操作

55.spring aop:@aspect 写在类上，@Before ，@After，@around，@afterthrowing

56.微服务架构：

a.配置中心

b.gateway

c.智能路由

d.复杂均衡：nacos自带和**Spring Cloud Load Balancer**

e.断路器