## 1、Java的内存模型以及GC算法

内存模型：程序计数器 线程私有，标记线程当前执行的位置

虚拟机栈 局部变量，操作数栈、方法出口、动态链接

本地方法栈 native方法

堆 对象实例和数组

方法区 被JVM加载的类信息，常量，静态变量，即时编译器编译后的代码

GC算法： 复制算法 将内存容量划分成两块，每次只使用其中一块，当该块内存用完时，将存活的对象复制到另一块内存并清除已用过的内存

标记清除 标记要回收的对象，标记完成后统一做清除（容易造成不连续的内存碎片）

标记整理 标记要回收的对象，标记完成后让所有存活的对象向一端移动，清理掉端边界以外的内存

## 2、Java 的引用类型有哪几种

强引用：有引用变量指向时永远不会被回收

软引用：有用但非必须，在内存不足时被回收

弱引用：每次垃圾回收都会被回收

虚引用：跟踪对象被回收的状态

3、Java 中 WeakReference 与 SoftReference的区别

4、Jvm性能调优都做了什么

选择合适的gc算法；调整堆栈的参数

5、Java的内存管理

6、介绍JVM中7个区域，每个区域可能造成内存的溢出的情况

程序计数器：不会溢出

堆溢出 OOM-java heap space List无限添加对象

虚拟机栈溢出 StackOverflowError：请求的栈深度>虚拟机允许的深度，如不停的i++；（调大栈容量-Xss）

OOM：不停的创建新线程（可以通过减少最大堆及减少栈容量；或减少线程数解决）

本地方法栈溢出 同上（只是执行的是本地C/C++方法）

方法区溢出：OutOfMemoryError:PermGen space 加载过多class文件(jar文件)-动态代理，j sp，OSGI，

运行时常量溢出 OutOfMemoryError:PermGen space 如使用jkd1.6及之前版本String.intern模拟 1.6及以前把首次遇到的字符串实例复制到字符串常量池，1.7是在常量池中记录首次出现的实例引用 -XX:PermSize

直接内存溢出 JVM所需内存 + 直接内存 > 机器物理内存(或操作系统级限制)，无法动态拓展 如使用了NIO（Buffer提供了一种直接操作操作系统缓冲区的方式，即ByteBuffer.allocateDirector(size)，这个方法返回DirectByteBuffer应用就是指向这个底层存储空间关联的缓冲区，即直接内存） -XX：MaxDirectMemorySize

## 7、垃圾回收机制 分代回收

垃圾回收机制：先采用可达性分析算法确定对象是否存活，采用分代回收将非存活的对象回收

分代回收：堆区根据对象存活的生命周期将内存划分新生代跟老年代；方法区-永久代

新生代：存放生命周期较短的对象，Eden区及两个survivor区，当eden区满时采用复制算法将存活的对象移至其中一个survivor区，当这块survivor也满是将存活对象复制到另一块survivor，两个survivor交替工作，到达一定条件的对象回进入老年代

老年代：存放生命周期较长的对象，大对象（如长字符串，数组）；采用标记清除/标记整理算法进行回收；当老年代的连续空间小于新生代对象总大小或者小于历次晋升的平均大小会触发Full GC

1、Java的基础类型以及包装类

2、Integer的缓存机制

IntegerCache缓存了-128到127，当在该范围内int自动转成Integer时，会直接存缓存中找

3、String类为什么是final的

final的出现就是为了为了不想改变，而不想改变的理由有两点：设计(安全)或者效率。

final 修饰的类是不被能继承的，所以 final 修饰的类是不能被篡改的

要保证 java.lang.String 引用的对象一定是 java.lang.String的对象，而不是引用它的子孙类，

这样才能保证它的效率和安全

4、String、StringBuilder、StringBuffer区别

String 字符串常量；是不可变的对象, 因此在每次对 String 类型进行改变的时候其实都等同

于生成了一个新的 String 对象，然后将指针指向新的 String 对象

StringBuffer 字符串变量；每次结果都会对 StringBuffer 对象本身进行操作，而不是生成新的

对象，再改变对象引用

StringBuilder 同上 但线程不安全，适用于单线程

5、String s="123"  和String s=new String(“123”)区别

s=”123”创建一个对象放在常量池中

new String(“123”) 创建一个对象123放在常量池，创建一个对象放在堆区

6、String s=new String(“123”) 内存分布状态

在字符串常量池里分配一个对象123

New String在堆中创建一个对象

7、接口与抽象类的区别

第一点． 接口是抽象类的变体，接口中所有的方法都是抽象的。而抽象类是声明方法的存

在而不去实现它的类。

第二点． 接口可以多继承，抽象类不行

第三点． 接口定义方法，不能实现，而抽象类可以实现部分方法。

第四点． 接口中基本数据类型为static 而抽类象不是的。

8、hashcode()和equal()方法的区别

HashCode相当不代表equal相等；equal相等hashcode一定相等

1、 通过hashCode可以很快的查到小内存块。

2、 通过hashCode比较比equal方法快，当get时先比较hashCode，如果hashCode不同，

直接返回false

9、== 和 equals 的区别

==比较的是对象地址

equals比较的是值

10、哪个类包含 clone 方法？是 Cloneable 还是 Object

Object 创建并返回此对象的一个副本

11、深度拷贝与浅拷贝的区别

浅拷贝只克隆他自身以及他包含的所有对象的引用地址（object.clone是浅拷贝）

深拷贝拷贝除自身以外所有的对象，包括自身所包含的所有**对象实例（序列化是一种深拷贝）**

12、final、finalize 和 finally 的区别

Final是关键字，修饰符。用于声明属性，方法和类，分别表示属性不可交变，方法不可覆盖，

类不可继承。

finally是异常处理语句结构的一部分，表示总是执行。

finalize是Object类的一个方法，在垃圾收集器执行的时候会调用被回收对象的此方法，供

垃圾收集时的其他资源回收，例如关闭文件等。

13、类加载机制，有什么好处；如何实现类的隔离；类加载的过程，类加载不到报什么异常

好处：将Class字节码解析成JVM统一要求的格式？？

实现类的隔离：将自定义的classloader不要放在classpath下

ClassNotFoundException

14、Java异常体系  什么是受检异常，什么是运行时异常，throw 和 throws 有什么区别

受检的异常在编译的时候就会检查要求对它进行显式的try..catch 捕获处理或者向上一层

方法抛出,否则在编译期间就显示错误

运行时异常不需要强制处理所有的RuntimeException的子类都是运行时异常

1)throw 是手动抛出异常,throw new \*\*Exception(); 抛出的是某一个异常类型的实例

2)throws 是方法抛出异常,写在方法声明处

15、注解的底层实现原理

通过反射实现 实现了InvocationHandler

16、枚举的使用场景，以及底层实现原理

只要是需要控制变量的数量或者范围，并且拿到变量后还需要处理一些逻辑的场景都可以用

枚举来完成

枚举本质上是通过普通的类来实现的，只是编译器为我们进行了处理。每个枚举类型都继承

自java.lang.Enum，并自动添加了values和valueOf方法。而每个枚举常量是一个静态常量

字段，使用内部类实现，该内部类继承了枚举类。所有枚举常量都通过静态代码块来进行初

始化，即在类加载期间就初始化。另外通过把clone、readObject、writeObject这三个方法定

义为final的，同时实现是抛出相应的异常。这样保证了每个枚举类型及枚举常量都是不可

变的。可以利用枚举的这两个特性来实现线程安全的单例。

1、Java中数组的底层实现，如何高效的判断数组中是否包含某个元素

数组是一种引用数据类型，数组引用变量只是一个引用，数组元素和数组变量在内存里是分

开存放的, 而实际的数组元素被存储在堆(Heap)内存中，引用变量则被存储在栈(Stack)内存

中

循环遍历数组

2、Collections和Collection有什么差别

Collections是个java.util下的类，它包含有各种有关集合操作的静态方法。

Collection是个java.util下的接口，它是各种集合结构的父接口

3、Collection接口和Map接口的区别

4、描述一下ArrayList和LinkedList各自实现和区别，使用场景

ArrayList是基于数组实现的

LinkedList是基于链表

当只是需要在末尾增加或者删除元素，或者查询比较频繁时建议用ArrayList

当在指定位置添加或删除元素，而对查询效率要求较低或者按照顺序来查询时用LinkedList

5、ArrayList的底层结构，扩容原理；相关方法: remove sublist iterator

ArrayList底层是一个Object数组 默认初始容量=10

6、LinkedList 是单向链表还是双向链表

7、Set和List的区别

Set 元素添加无序不可重复（但是元素在set中的位置是由该元素的HashCode决定的）

List 元素添加有序可重复

8、HashSet与TreeSet的区别，使用场景

9、HashMap实现原理，底层结构

10、Hashtable和HashMap的区别，使用场景

11、JDK包下的Properties实现原理

12、ConcurrentHashMap的底层结构，读方法加锁吗? 如何实现了读的时候不加锁

13、TreeMap的实现原理

14、Comparator和Comparable接口的区别

1、字节流、字符流

字符流处理的单元为2个字节的Unicode字符，分别操作字符、字符数组或字符串

而字节流处理单元为1个字节，操作字节和字节数组

2、什么时候使用字节流、什么时候使用字符流?

3、如何在字符流和字节流之间转换

InputStreamReader和OutputStreamWriter，这两个为类是字节流和字符流之间相互转换的

类

4、递归读取文件夹下的文件，代码怎么实现

Java创建线程之后，直接调用start()方法和run()的区别

sleep()和wait()区别

sleep让调用线程进入睡眠，让出执行机会给其他线程，等休眠结束，线程进入就绪状态和

其他线程一起竞争。Sleep是Thread的静态方法，他不会改变对象的锁，被线程持有的锁不

会释放

Wait：Object的方法，线程执行到wait时，进入到一个与改对象相关的等待池，同时释放

锁，是其他线程能够访问

wait()和notify()方法使用场景，各自含义

写一个死锁的程序

现在有T1、T2、T3三个线程，你怎样保证T2在T1执行完后执行，T3在T2执行完后执行？

什么是守护线程，守护线程和非守护线程的区别以及用法

线程池的原理，各大参数的含义；IO密集型线程池应该如何选择，CPU密集型线程池如何选择，抛弃策略

synchronized原理

ReentrantLock的原理

可重入锁的含义，以及ReentrantLock 和synchronized的区别

场景：在一个主线程中，要求有大量(很多很多)子线程执行完之后，主线程才执行完成。多种方式，考虑效率

定时线程的使用

BlockingQueue的使用。（take，poll的区别，put，offer的区别）

高并发情况下，系统是如何支撑大量的请求的

如果有一个特别大的访问量，到数据库上，怎么做优化（DB设计，DB IO，SQL优化，Java优化）

如果出现大面积并发，在不增加服务器的基础上，如何解决服务器响应不及时问题

假如你的项目出现性能瓶颈了，你觉得可能会是哪些方面，怎么解决问题

Threadlocal原理，Threadlocal使用完不释放会出现什么问题

Volatile的原理，volatile修饰数组 数组元素线程安全吗

在Java中CycliBarriar和CountdownLatch有什么区别？

Executor和Executors的区别

同步和异步，阻塞和非阻塞

Buffer：flip方法  clear方法  rewind方法

java序列化，自定义序列化方式如何实现，序列化 ID 的作用

Transient 关键字的作用

NIO AIO BIO

Reactor模式，Proactor模式

简单设计一个nio程序

Netty（线程模型，零拷贝，里面涉及的设计模式，心跳的实现）

nio中select方法是阻塞的吗

说说浏览器访问 ，经历了怎样的过程 www.baidu.com

Servlet 生命周期

Servlet对象是单实例？Servlet线程安全的？

Servlet中，filter的应用场景有哪些？采用了什么设计模式

http get和post区别

redirect与forward做跳转有什么区别？

web应用服务器对用户请求通常返回一些状态码，请描述下分别以4和5开头的状态码

session和cookie的区别和联系，session的生命周期

SpringMVC的原理

MVC:用户—>controller（接收请求，调用业务类）->交给Model层（service，dao，entity）->controller(派发页面->View)->相应用户

客户端将http请求给web服务器，web服务器对http请求进行解析，匹配web容器将请求

转交给dispatcherServlert，ds根据请求信息+HandlerMapping找到处理请求的处理器handler，

handler对数据处理完毕后返回一个modelAndView给ds，ds通过viewReoler将逻辑视图转

为真正的视图，ds将model解析出视图需要的参数将最终的完整视图返回给客户端

Spring中BeanFactory和ApplicationContext的联系和区别

ApplicationContext是具有BeanFactory的所有功能，待提供国际化消息访问、资源访问，事

件传播等

BeanFactory是只有在getBean时才加载并实例化Bean，配置中如果有其他的配置错误不会影响程序运行

ApplicationContext在容器启动时把xml配置读入内存进行校验并主动创建所有bean，

事务隔离级别

Read uncommitted读未提交 脏读；（可以读取另一个事务未提交的数据）

Read committed 读已提交（等到另一个事务提交后才能读取） 可以解决脏读 ，但会引起不可重复读，在两次查询发现结果不一致

Repeatable read 重复读（读事务开启后不允许update，但是允许insert）

解决不可重复读，存在幻读（对应insert）

Serializable 序列化 最高隔离级别 事务串行化顺序执行

Mysql 默认重复读 sqlServer Oracle默认读已提交

Spring如何实现事务管理的

编码方式；transactionTemplate.execute

声明式事务管理方式：基于AOP技术在方法执行前后进行拦截，然后在目标方法开始之前创建并加入事务，执行完目标方法后根据执行情况提交或回滚事务

1. TransactionProxyFactoryBean 需要为每个进行事务管理的类做相应配置
2. 基于AspectJ的XML方式，不需要改动类，在XML文件中配置好即可
3. 基于注解@Transactional的方式，配置简单，需要在业务层类中添加注解

Spring的事务传播机制

spring事务开启 ，或者是基于接口的或者是基于类的代理被创建，所以在同一个类中一个

方法调用另一个方法有事务的方法，事务是不会起作用的

PROPAGATION\_REQUIRED -- 支持当前事务，如果当前没有事务，就新建一个事务。这是最常见的选择。 （如果当前有事务，出现异常 一起回滚）

PROPAGATION\_SUPPORTS -- 支持当前事务，如果当前没有事务，就以非事务方式执行。

PROPAGATION\_MANDATORY -- 支持当前事务，如果当前没有事务，就抛出异常。

PROPAGATION\_REQUIRES\_NEW -- 新建事务，如果当前存在事务，把当前事务挂起。

（新事物不依赖于当前事务的commit跟rolled back。当前有事务，如果当前事务失败，新事务中已经提交的不会被回滚；新事务提交抛异常，如果被当前事务捕获则当前事务仍可以提交，否则当前事务将回滚）

PROPAGATION\_NOT\_SUPPORTED -- 以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。

PROPAGATION\_NEVER -- 以非事务方式执行，如果当前存在事务，则抛出异常。

PROPAGATION\_NESTED -- 如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则执行与 PROPAGATION\_REQUIRED 类似的操作

Spring IoC

Spring自定义扩展标签如何实现

1：编写一个xml架构来描述自定义的元素

2：编写一个自定义的handler实现NamespaceHandler接口

3：编写一个或者多个自定义的解析器实现BeanDefinitionParser接口

4：注册到Spring容器

Spring中InitializingBean接口的作用

spring初始化bean的时候，如果bean实现了InitializingBean接口，会自动调用

afterPropertiesSet方法。

spring为bean提供了两种初始化bean的方式：（实现InitializingBean接口，实现

afterPropertiesSet方法，或者在配置文件中同过init-method指定，两种方式可以同时使用）

Spring AOP的原理   jdk动态代理实现原理  cglib和jdk动态代理的区别

Aop代理分成静态代理（aspectJ）和动态代理（spring aop）

静态代理就是aop框架会在编译阶段生成aop代理类（编译时增强）

aspectJ：在编译阶段将aspect织入java字节码，运行时就是经过增强后的aop对象

public aspect TxAspect {

    void around():call(void Hello.sayHello()){

        System.out.println("开始事务 ...");

        proceed();// 执行被代理类中的方法

        System.out.println("事务结束 ...");

    }

}

Spring aop采用动态代理，不会去修改字节码，而是在内存中临时为方法生成一个aop对象，aop对象包含了目标对象的全部方法，并在特定切点做了增强处理，并回调原对象方法

JDK：通过反射接收被代理类，被代理类必须实现一个接口； invocationHandler接口+Proxy类 通过Proxy.newProxyInstance生成被代理对象，在invocationHandler.invoke方法实现中可以在需要执行的方法前后加入我们需要统一处理的逻辑

1. 根据classloader+interfaces创建代理类 实现interfaces接口并继承了Proxy类）
2. 实例化代理类，在构造方法中传入被代理对象，代理类调用父类Proxy构造器为invocationHandler赋值
3. 生成的代理类，在调用方法时，会调用Proxy的Handler的invoke方法

Cglib：可以对没有实现接口的类进行代理。在运行时通过字节码技术动态生成被代理类的子类，通过继承的方式实现（方法不能被定义为final）：Enhancer+MothodIntercepter

Enhancer.create() 生成被代理类的子类

介绍常用的数据结构；redis事务；发布订阅功能

String、list、set、sorted set、hash （memcached只支持String，不能持久化数据）

执行一组指令时，当中有失败的，前面也不回滚

发布订阅基于channel 或者根据某种模式匹配channel

MySQL里有2000w数据，redis中只存20w的数据，如何保证redis中的数据都是热点数据=》回收策略

设置过期时间，每次对命中缓存的数据增加一定的过期时间

* 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

主从同步方案

全量同步：一般发生在slave初始化阶段。S连接m，发送SYNC命令，m接收命令执行bgsave命令生成rdb文件并使用缓冲区记录伺候执行的所有写命令；m在bgsave完后向s发送快照文件，s收到快照文件后丢弃所有旧数据，载入快照；m快照发送完毕后向从服务器发送缓冲区的写命令，s完成快照载入后开始接收命令执行m的写命令

增量同步：主服务器每执行一个写命令就会向从服务器发送相同的写命令，从服务器接收并执行收到的写命令

备份方案

定期生成rbd文件，根据小时、天等保存到文件夹，对rdb文件进行复制

字典类型通过什么方式rehash

事务如何实现

redis的线程模型，为什么是单线程

RDB与AOF区别

Rdb：可以在指定的时间间隔内生成数据集的时间点快照，是一个二进制存储文件

Aof记录服务器执行的所有写命令，会被追加到文件的末尾，以redis协议的格式保存

数据库事务特性

事务的特性：ACID

原子性：一个事务中的操作要么全做，要么全不做

一致性：数据不会因为事务的执行而破坏

隔离性：一个事务的执行不受其他事务干扰

持久性：事务一旦提交，它对数据库的改变是永久的

数据库范式(需要再消化)

第一范式：无重复的列

第二范式：数据库表中的每个实例或行必须可以被唯一地区分（要有唯一标志）

第三范式：….

主从一致性如何保证

主从复制原理：master将事务串行写入二进制日志，完成后通知存储引起提交事务。在slave

上会有I/O线程跟SQL线程，IO线程在master上打来一个普通的链接，开始binlog dump

Process，他会从master的二进制日志中读取事件，将事件写入relay log。Sql线程从relay log

中读取事件，重放其中的事件来更新slave数据从而与master数据保持一致

1、Master上修改配置：

*innodb\_flush\_log\_at\_trx\_commit = 1*

*sync\_binlog = 1*

保证每次事务提交后，都能实时刷新到磁盘中，尤其是确保每次事务对应的binlog都能及

时刷新到磁盘中，只要有了binlog，InnoDB就有办法做数据恢复，不至于导致主从复制的

数据丢

2、slave上修改配置：

*master\_info\_repository = "TABLE"*

*relay\_log\_info\_repository = "TABLE"*

*relay\_log\_recovery = 1*

开启relay log自动修复机制，发生crash时，会自动判断哪些relay log需要重新从master

上抓取回来再次应用，以此避免部分数据丢失的可能性

db中大字段如何处理（读为主 or 写为主）

innodb是行锁还是表锁？如何实现

行锁。行锁是加在索引响应行上的，如果对应的sql没有走索引，那么会全表扫描（表锁）

**表锁**：不会出现死锁，发生锁冲突几率高，并发低。

**行锁：**会出现死锁，发生锁冲突几率低，并发高。分为共享锁（读锁）和排他锁（写锁）

共享锁是我读时你也能读但不能写；排他锁是我写时你不能写也不能读

insert ，delete ， update在事务中都会自动默认加上排它锁

innodb存储引擎内存结构

**innodb\_buffer\_pool**

**innodb附加内存池**

**innodb日志缓冲**

innodb中索引工作原理和种类

如果是主键B+数从根节点遍历分支，找到对应的叶节点，获得行数据

如果是辅索引B+数 找到对应的叶节点获得主键，在用主键在主键B+数中检索

innodb如何实现事务

隔离性由锁实现

原子性和持久性---redo log（事务日志）：提交事务时，innodb存储引擎把事务写入日志缓冲，日志缓冲把事务刷新到事务日志；innodb把事务写入缓冲池

一致性:undo log逻辑日志，回滚行记录到某个特定版本

RocketMQ:

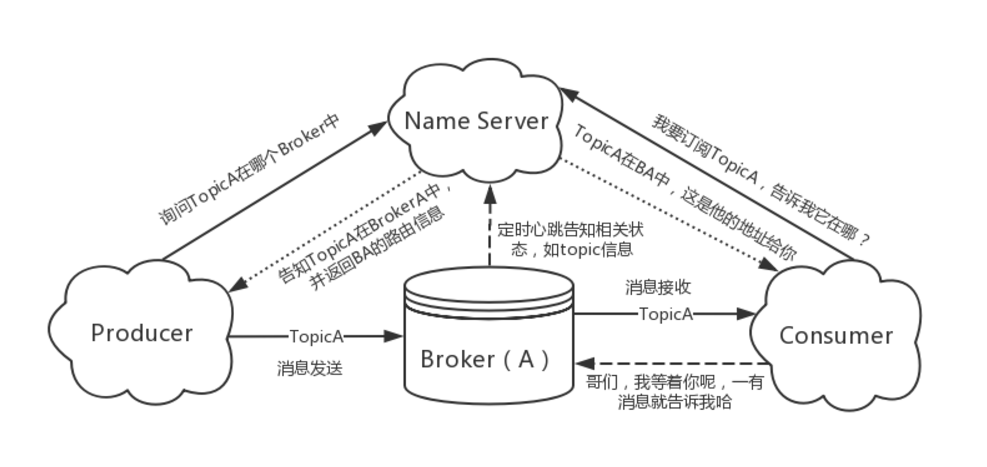
nameServer：无状态节点，保存broker列表和topic列表

broker：负责存储消息及转发消息（控制topic的增删及订阅）

工作流程：

1. 搭建nameserver，producer、consumer和broker之间的联系通过访问nameserver获取
2. 启动broker进行topic的增加和订阅
3. Procuder，consumer定期从nameserver获取路由信息

4、producer发送已有的topic消息到broker，consumer从broker拉取消息



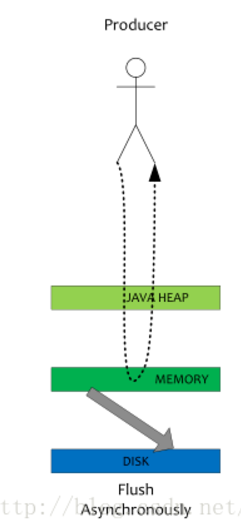
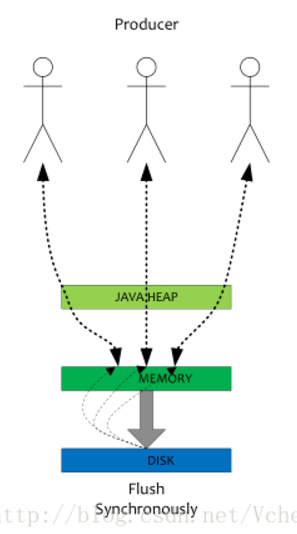
MQ

主从如何备份

在master上维护一个变量，将该变量设为对应的offset，不停的查看slave上报的进度是否超过这个进度，超过则写入成功

文件在磁盘上面如何存储

Rocketmq的所有消息都是持久化的，先写入系统pagecache，然后刷盘

异步刷盘：写完pagecache唤醒刷盘线程后直接返回（FlushRealTimeService）

同步刷盘：等待刷盘完成才能返回（GroupCommitService）

数据不丢失是如何保证的？

消息顺序是如何保证的？

真正使用过程中对mq的使用场景，有哪些需要注意的点

消息堆积？重复消费？

ES

1、ES中query和match的区别

2、在并发情况下，Elasticsearch如果保证读写一致

3、数据插入到ES集群到存储完成的过程

4、数据从查询api调用到返回数据的过程

5、ES更新和删除文档的过程

6、master选举的流程

7、脑裂问题如何避免

当集群中master候选的个数不小于3个（node.master: true）。可以通过

discovery.zen.minimum\_master\_nodes 这个参数的设置来避免脑裂，设置为(N/2)+1

当=2时将一个node.master改成false

Dubbo中zookeeper做注册中心，如果注册中心集群都挂掉，发布者和订阅者之间还能通信么

Dubbo降级如何实现

Dubbo流控如何实现

Dubbo负载均衡策略

Dubbo是如何运用SPI机制的

Dubbo中filter的作用，这是一种什么设计模式

Dubbo在安全机制方面是如何解决的

Dubbo的架构

Dubbo服务注册与发现的流程

Dubbo的容灾和负载如何实现

Dubbo的网络层（netty）、协议层（hession序列化)，心跳机制是如何实现的

Dubbo客户端如何实现连接复用的

Dubbo客户端发起远程调用有几种方式，各自的区别

Dubbo远程调用采用异步方式，异步方式是一种future模式，这是什么模式

Dubbo的隐式传参和泛化调用

Dubbo优雅停机是怎么实现的

Dubbo中涉及的设计模式

两个有序数组的合并排序

 一个数组的倒序

 二叉树的遍历算法

 比较重要的数据结构，如链表，队列，栈的基本理解及大致实现。

 如果要25匹马中选出跑得最快的3匹，每次只有5匹马同时跑，最少要比赛几次，才能确保得出结果？