一、java 基础篇

1. 接口和抽象类的区别

相似点：

（1）接口和抽象类都不能被实例化

（2）实现接口或继承抽象类的普通子类都必须实现这些抽象方法

不同点：

（1）抽象类可以包含普通方法和代码块，接口里只能包含抽象方法，静态方法和默认方法，

（2）抽象类可以有构造方法，而接口没有

（3）抽象类中的成员变量可以是各种类型的，接口的成员变量只能是 public static final 类型的，并且必须赋值

2. 重载和重写的区别

重载发生在同一个类中，方法名相同、参数列表、返回类型、权限修饰符可以不同

重写发生在子类中，方法名相、参数列表、返回类型都相同，权限修饰符要大于父类方法，声明异常范围要小于父类方法，但是 final 和 private 修饰的方法不可重写

3. equals 和==的区别

==比较基本类型，比较的是值，==比较引用类型，比较的是内存地址

equlas 是 Object 类的方法，本质上与==一样，但是有些类重写了 equals 方法，比如 String 的 equals 被重写后，比较的是字符值，另外重写了 equlas 后，也必须重写 hashcode()方法

4. 异常处理机制

（1）使用 try、catch、finaly 捕获异常，finaly 中的代码一定会执行，捕获异常后程序会继续执行

（2）使用 throws 声明该方法可能会抛出的异常类型，出现异常后，程序终止

5. HashMap 原理

(1).HashMap 在 Jdk1.8 以后是基于数组+链表+红黑树来实现的，特点是，key 不能重复，可以为 null，线程不安全

(2).HashMap 的扩容机制：

HashMap 的默认容量为 16，默认的负载因子为 0.75，当 HashMap 中元素个数超过容量乘以负载因子的个数时，就创建一个大小为前一次两倍的新数组，再将原来数组中的数据复制到新数组中。当数组长度到达 64 且链表长度大于 8 时，链表转为红黑树

(3).HashMap 存取原理：

1）计算 key 的 hash 值，然后进行二次 hash，根据二次 hash 结果找到对应的索引位置

2）如果这个位置有值，先进性 equals 比较，若结果为 true 则取代该元素，若结果为 false，就使用高低位平移法将节点插入链表（JDK8 以前使用头插法，但是头插法在并发扩容时可能会造成环形链表或数据丢失，而高低位平移发会发生数据覆盖的情况）

6. 想要线程安全的 HashMap 怎么办？

（1）使用 ConcurrentHashMap

（2）使用 HashTable

（3）Collections.synchronizedHashMap()方法

7. ConcurrentHashMap 原如何保证的线程安全？

JDK1.7:使用分段锁，将一个 Map 分为了 16 个段，每个段都是一个小的 hashmap，每次操作只对其中一个段加锁

JDK1.8:采用 CAS+Synchronized 保证线程安全，每次插入数据时判断在当前数组下标是否是第一次插入，是就通过 CAS 方式插入，然后判断 f.hash 是否=-1，是的话就说明其他线程正在进行扩容，当前线程也会参与扩容；删除方法用了 synchronized 修饰，保证并发下移除元素安全

8. HashTable 与 HashMap 的区别

（1）HashTable 的每个方法都用 synchronized 修饰，因此是线程安全的，但同时读写效率很低

（2）HashTable 的 Key 不允许为 null

（3）HashTable 只对 key 进行一次 hash，HashMap 进行了两次 Hash

（4）HashTable 底层使用的数组加链表

9. ArrayList 和 LinkedList 的区别

ArratList 的底层使用动态数组，默认容量为 10，当元素数量到达容量时，生成一个新的数组，大小为前一次的 1.5 倍，然后将原来的数组 copy 过来；因为数组在内存中是连续的地址，所以 ArrayList 查找数据更快，由于扩容机制添加数据效率更低

LinkedList 的底层使用链表，在内存中是离散的，没有扩容机制；LinkedList 在查找数据时需要从头遍历，所以查找慢，但是添加数据效率更高

10.如何保证 ArrayList 的线程安全？

（1）使用 collentions.synchronizedList（）方法为 ArrayList 加锁

（2）使用 Vector，Vector 底层与 Arraylist 相同，但是每个方法都由 synchronized 修饰，速度很慢

（3）使用 juc 下的 CopyOnWriterArrayList，该类实现了读操作不加锁，写操作时为 list 创建一个副本，期间其它线程读取的都是原本 list，写操作都在副本中进行，写入完成后，再将

指针指向副本。

11. String、StringBuffer、StringBuilder 的区别

String 由 char[] 数组构成，使用了 final 修饰，对 String 进行改变时每次都会新生成一个 String 对象，然后把指针指向新的引用对象。

StringBuffer 可变并且线程安全

StringBuiler 可变但线程不安全。

操作少量字符数据用 String；单线程操作大量数据用 StringBuilder；多线程操作大量数据用 StringBuffer。

12. hashCode 和 equals

hashCode()和 equals()都是 Obkect 类的方法，hashCode()默认是通过地址来计算 hash 码，但是可能被重写过用内容来计算 hash 码，equals()默认通过地址判断两个对象是否相等，但是可能被重写用内容来比较两个对象

所以两个对象相等，他们的 hashCode 和 equals 一定相等，但是 hashCode 相等的两个对象未必相等

如果重写 equals()必须重写 hashCode()，比如在 HashMap 中，key 如果是 String 类型，String 如果只重写了 equals（）而没有重写 hashcode（）的话，则两个 equals()比较为 true 的 key，因为 hashcode 不同导致两个 key 没有出现在一个索引上，就会出现 map 中存在两个相同的 key

13. 面向对象和面向过程的区别

面向对象有封装、继承、多态性的特性，所以相比面向过程易维护、易复用、易扩展，但是因为类调用时要实例化，所以开销大性能比面向过程低

14.深拷贝和浅拷贝

浅拷贝:浅拷贝只复制某个对象的引用，而不复制对象本身，新旧对象还是共享同一块内存
深拷贝:深拷贝会创造一个一摸一样的对象，新对象和原对象不共享内存，修改新对象不会改变原对对象。

14.多态的作用

多态的实现要有继承、重写，父类引用指向子类对象。它的好处是可以消除类型之间的耦合关系，增加类的可扩充性和灵活性。

15.什么是反射？

反射是通过获取类的 class 对象，然后动态的获取到这个类的内部结构，动态的去操作类的属性和方法。
应用场景有：要操作权限不够的类属性和方法时、实现自定义注解时、动态加载第三方 jar 包时、按需加载类，节省编译和初始化时间；
获取 class 对象的方法有：class.forName(类路径)，类.class()，对象的 getClass（）

16.Java 创建对象得五种方式？

(1)new 关键字　(2)Class.newInstance　(3)Constructor.newInstance

(4)Clone 方法　(5)反序列化

二.Java 多线程篇

1. 进程和线程的区别，进程间如何通信

进程：系统运行的基本单位，进程在运行过程中都是相互独立，但是线程之间运行可以相互影响。

线程：独立运行的最小单位，一个进程包含多个线程且它们共享同一进程内的系统资源

进程间通过管道、 共享内存、信号量机制、消息队列通信

2. 什么是线程上下文切换

当一个线程被剥夺 cpu 使用权时，切换到另外一个线程执行

3. 什么是死锁

死锁指多个线程在执行过程中，因争夺资源造成的一种相互等待的僵局

4. 死锁的必要条件

互斥条件：同一资源同时只能由一个线程读取

不可抢占条件：不能强行剥夺线程占有的资源

请求和保持条件：请求其他资源的同时对自己手中的资源保持不放

循环等待条件：在相互等待资源的过程中，形成一个闭环

想要预防死锁，只需要破坏其中一个条件即可，比如使用定时锁、尽量让线程用相同的加锁顺序，还可以用银行家算法可以预防死锁

5. Synchrpnized 和 lock 的区别

（1）synchronized 是关键字，lock 是一个类

（2） synchronized 在发生异常时会自动释放锁，lock 需要手动释放锁

（3）synchronized 是可重入锁、非公平锁、不可中断锁，lock 的 ReentrantLock 是可重入锁，可中断锁，可以是公平锁也可以是非公平锁

（4）synchronized 是 JVM 层次通过监视器实现的，Lock 是通过 AQS 实现的

6. 什么是 AQS 锁?

AQS 是一个抽象类，可以用来构造锁和同步类，如 ReentrantLock，Semaphore，CountDownLatch，CyclicBarrier。

AQS 的原理是，AQS 内部有三个核心组件，一个是 state 代表加锁状态初始值为 0，一个是获取到锁的线程，还有一个阻塞队列。当有线程想获取锁时，会以 CAS 的形式将 state 变为 1，CAS 成功后便将加锁线程设为自己。当其他线程来竞争锁时会判断 state 是不是 0，不是 0 再判断加锁线程是不是自己，不是的话就把自己放入阻塞队列。这个阻塞队列是用双向链表实现的

可重入锁的原理就是每次加锁时判断一下加锁线程是不是自己，是的话 state+1，释放锁的时候就将 state-1。当 state 减到 0 的时候就去唤醒阻塞队列的第一个线程。

7.为什么 AQS 使用的双向链表?

因为有一些线程可能发生中断 ，而发生中断时候就需要在同步阻塞队列中删除掉，这个时候用双向链表方便删除掉中间的节点

7. 有哪些常见的 AQS 锁

AQS 分为独占锁和共享锁

ReentrantLock（独占锁）：可重入，可中断，可以是公平锁也可以是非公平锁，非公平锁就是会通过两次 CAS 去抢占锁，公平锁会按队列顺序排队

Semaphore（信号量）:设定一个信号量，当调用 acquire()时判断是否还有信号，有就获取一个信号量，没有就阻塞等待其他线程释放信号量，当调用 release()时释放一个信号量，唤醒阻塞线程。

应用场景：允许多个线程访问某个临界资源时，如上下车，买卖票

CountDownLatch（倒计数器）:给计数器设置一个初始值，当调用 CountDown()时计数器减一，当调用 await() 时判断计数器是否归 0，不为 0 就阻塞，直到计数器为 0。

应用场景：启动一个服务时，主线程需要等待多个组件加载完毕，之后再继续执行

CyclicBarrier（循环栅栏）:给计数器设置一个目标值,当调用 await() 时会计数+1 并判断计数

器是否达到目标值，未达到就阻塞，直到计数器达到目标值

应用场景：多线程计算数据，最后合并计算结果的应用场景

9.sleep()和 wait()的区别

(1)wait()是 Object 的方法，sleep()是 Thread 类的方法

(2)wait()会释放锁，sleep()不会释放锁

(3)wait()要在同步方法或者同步代码块中执行，sleep()没有限制

(4)wait()要调用 notify()或 notifyall()唤醒,sleep()自动唤醒

10.yield()和 join()区别

yield()调用后线程进入就绪状态

A 线程中调用 B 线程的 join() ,则 B 执行完前 A 进入阻塞状态

11.线程池七大参数

核心线程数：线程池中的基本线程数量

最大线程数：当阻塞队列满了之后，逐一启动

最大线程的存活时间：当阻塞队列的任务执行完后，最大线长的回收时间

最大线程的存活时间单位

阻塞队列：当核心线程满后，后面来的任务都进入阻塞队列

线程工厂：用于生产线程

任务拒绝策略：阻塞队列满后，拒绝任务，有四种策略（1）抛异常（2）丢弃任务不抛异常（3）打回任务（4）尝试与最老的线程竞争

12. Java 内存模型

JMM（Java 内存模型 ）屏蔽了各种硬件和操作系统的内存访问差异，实现让 Java 程序在各平台下都能达到一致的内存访问效果，它定义了 JVM 如何将程序中的变量在主存中读取

具体定义为：所有变量都存在主存中，主存是线程共享区域；每个线程都有自己独有的工作内存，线程想要操作变量必须从主从中 copy 变量到自己的工作区，每个线程的工作内存是相互隔离的

由于主存与工作内存之间有读写延迟，且读写不是原子性操作，所以会有线程安全问题

13. 保证并发安全的三大特性？

原子性：一次或多次操作在执行期间不被其他线程影响

可见性：当一个线程在工作内存修改了变量，其他线程能立刻知道

有序性：JVM 对指令的优化会让指令执行顺序改变，有序性是禁止指令重排

14. Volatile

保证变量的可见性和有序性，不保证原子性。使用了 volatile 修饰变量后，在变量修改后会立即同步到主存中，每次用这个变量前会从主存刷新。

单例模式双重校验锁变量为什么使用 volatile 修饰？ 禁止 JVM 指令重排序，new Object() 分为三个步骤：为实例对象分配内存，用构造器初始化成员变量，将实例对象引用指向分配的内存；实例对象在分配内存后实才不为 null。如果分配内存后还未初始化就先将实例对象指向了内存，那么此时最外层的 if 会判断实例对象已经不等于 null 就直接将实例对象返回。而此时初始化还没有完成。

15. 线程使用方式

(1)继承 Tread 类

(2)实现 Runnable 接口

(3)实现 Callable 接口：带有返回值

(4)线程池创建线程

16.ThreadLocal 原理

原理是为每个线程创建变量副本，不同线程之间不可见，保证线程安全。每个线程内部都维护了一个 Map，key 为 threadLocal 实例，value 为要保存的副本。
但是使用 ThreadLocal 会存在内存泄露问题，因为 key 为弱引用，而 value 为强引用，每次 gc 时 key 都会回收，而 value 不会被回收。所以为了解决内存泄漏问题，可以在每次使用完后删除 value 或者使用 static 修饰 ThreadLocal，可以随时获取 value

17.什么是 CAS 锁

CAS 锁可以保证原子性，思想是更新内存时会判断内存值是否被别人修改过，如果没有就直接更新。如果被修改，就重新获取值，直到更新完成为止。这样的缺点是

（1）只能支持一个变量的原子操作，不能保证整个代码块的原子操作

（2）CAS 频繁失败导致 CPU 开销大

（3）ABS 问题:线程 1 和线程 2 同时去修改一个变量，将值从 A 改为 B，但线程 1 突然阻塞，此时线程 2 将 A 改为 B,然后线程 3 又将 B 改成 A,此时线程 1 将 A 又改为 B,这个过程线程 2 是不知道的，这就是 ABA 问题，可以通过版本号或时间戳解决

18.Synchronized 锁原理和优化

Synchronize 是通过对象头的 markwordk 来表明监视器的，监视器本质是依赖操作系统的互斥锁实现的。操作系统实现线程切换要从用户态切换为核心态，成本很高，此时这种锁叫重量级锁，在 JDK1.6 以后引入了偏向锁、轻量级锁、重量级锁

偏向锁：当一段代码没有别的线程访问，此时线程去访问会直接获取偏向锁

轻量级锁：当锁是偏向锁时，有另外一个线程来访问，会升级为轻量级锁。线程会通过 CAS 方式获取锁，不会阻塞，提高性能，

重量级锁：轻量级锁自旋一段时间后线程还没有获取到锁，会升级为重量级锁，重量级锁时，来竞争锁的所有线程都会阻塞，性能降低

注意，锁只能升级不能降级

19.如何根据 CPU 核心数设计线程池线程数量

IO 密集型：线程中十分消耗 Io 的线程数*2
CPU 密集型： cpu 线程数量

20.AtomicInteger 的使用场景

AtomicInteger 是一个提供原子操作的 Integer 类，使用 CAS+volatile 实来现线程安全的数值操作。

因为 volatile 禁止了 jvm 的排序优化,所以它不适合在并发量小的时候使用，只适合在一些高并发程序中使用

三.JVM 篇

1. JVM 运行时数据区（内存结构）

线程私有区：

（1）虚拟机栈：每次调用方法都会在虚拟机栈中产生一个栈帧，每个栈帧中都有方法的参数、局部变量、方法出口等信息，方法执行完毕后释放栈帧

（2）本地方法栈：为 native 修饰的本地方法提供的空间，在 HotSpot 中与虚拟机合二为一

（3）程序计数器：保存指令执行的地址，方便线程切回后能继续执行代码

线程共享区：

（4）堆内存：Jvm 进行垃圾回收的主要区域，存放对象信息，分为新生代和老年代，内存比例为 1：2，新生代的 Eden 区内存不够时时发生 MinorGC,老年代内存不够时发生 FullGC

（5）方法区：存放类信息、静态变量、常量、运行时常量池等信息。JDK1.8 之前用持久代实现，JDK1.8 后用元空间实现，元空间使用的是本地内存，而非在 JVM 内存结构中

2. 什么情况下会内存溢出？

堆内存溢出：（1）当对象一直创建而不被回收时（2）加载的类越来越多时（3)虚拟机栈的线程越来越多时

栈溢出：方法调用次数过多，一般是递归不当造成

3. JVM 有哪些垃圾回收算法？

（1）标记清除算法： 标记不需要回收的对象，然后清除没有标记的对象，会造成许多内存碎片。
（2）复制算法： 将内存分为两块，只使用一块，进行垃圾回收时，先将存活的对象复制到另一块区域，然后清空之前的区域。用在新生代
（3）标记整理算法： 与标记清除算法类似，但是在标记之后，将存活对象向一端移动，然后清除边界外的垃圾对象。用在老年代

4. GC 如何判断对象可以被回收？

（1）引用计数法：已淘汰，为每个对象添加引用计数器，引用为 0 时判定可以回收，会有两个对象相互引用无法回收的问题

（2)可达性分析法：从 GCRoot 开始往下搜索，搜索过的路径称为引用链，若一个对象 GCRoot 没有任何的引用链，则判定可以回收

GCRoot 有：虚拟机栈中引用的对象，方法区中静态变量引用的对象，本地方法栈中引用的对象

5. 典型垃圾回收器

CMS:以最小的停顿时间为目标、只运行在老年代的垃圾回收器，使用标记-清除算法，可以并发收集。

G1 ：JDK1.9 以后的默认垃圾回收器，注重响应速度，支持并发，采用标记整理+复制算法回收内存，使用可达性分析法来判断对象是否可以被回收。

6. 类加载器和双亲委派机制

从父类加载器到子类加载器分别为：

BootStrapClassLoader　　加载路径为：JAVA_HOME/jre/lib

ExtensionClassLoader　　加载路径为：JAVA_HOME/jre/lib/ext

ApplicationClassLoader　加载路径为：classpath

还有一个自定义类加载器

当一个类加载器收到类加载请求时，会先把这个请求交给父类加载器处理，若父类加载器找不到该类，再由自己去寻找。该机制可以避免类被重复加载，还可以避免系统级别的类被篡改

7. JVM 中有哪些引用？

强引用：new 的对象。哪怕内存溢出也不会回收

软引用：只有内存不足时才会回收

弱引用：每次垃圾回收都会回收

虚引用：必须配合引用队列使用，一般用于追踪垃圾回收动作

8. 类加载过程

（1）加载 ：把字节码通过二进制的方式转化到方法区中的运行数据区

（2）连接：

验证：验证字节码文件的正确性。

准备：正式为类变量在方法区中分配内存，并设置初始值，final 类型的变量在编译时已经赋值了

解析：将常量池中的符号引用（如类的全限定名）解析为直接引用（类在实际内存中的地址）

（3）初始化 ：执行类构造器（不是常规的构造方法），为静态变量赋初值并初始化静态代码块。

9. JVM 类初始化顺序

父类静态代码块和静态成员变量->子类静态代码块和静态成员变量->父类代码块和普通成员变量->父类构造方法->子类代码块和普成员变量->子类构造方法

10.对象的创建过程

（1）检查类是否已被加载，没有加载就先加载类

（2）为对象在堆中分配内存，使用 CAS 方式分配，防止在为 A 分配内存时，执行当前地址的指针还没有来得及修改，对象 B 就拿来分配内存。

（3）初始化，将对象中的属性都分配 0 值或 null

（4）设置对象头

（5）为属性赋值和执行构造方法

11.对象头中有哪些信息

对象头中有两部分，一部分是 MarkWork,存储对象运行时的数据，如对象的 hashcode、GC 分代年龄、GC 标记、锁的状态、获取到锁的线程 ID 等；另外一部分是表明对象所属类，如果是数组，还有一个部分存放数组长度

12.JVM 内存参数

-Xmx[]:堆空间最大内存

-Xms[]:堆空间最小内存，一般设置成跟堆空间最大内存一样的

-Xmn[]:新生代的最大内存

-xx:[survivorRatio=3]:eden 区与 from+to 区的比例为 3：1，默认为 4：1

-xx[use 垃圾回收器名称]：指定垃圾回收器

-xss:设置单个线程栈大小

一般设堆空间为最大可用物理地址的百分之 80

13.GC 的回收机制和原理

GC 的目的实现内存的自动释放，使用可达性分析法判断对象是否可回收，采用了分代回收思想，
将堆分为新生代、老年代，新生代中采用复制算法，老年代采用整理算法，当新生代内存不足时会发生 minorGC,老年代不足时会发送 fullGC

四.Mysql 篇

1. MyIsAm 和 InnoDB 的区别

InnoDB 有三大特性，分别是事务、外键、行级锁，这些都是 MyIsAm 不支持的，

另外 InnoDB 是聚簇索引，MyIAm 是非聚簇索引，

InnoDB 不支持全文索引，MyIAm 支持

InnoDB 支持自增和 MVCC 模式的读写，MyIAm 不支持

MyIsAM 的访问速度一般 InnoDB 快，差异在于 innodb 的 mvcc、行锁会比较消耗性能，还可

能有回表的过程（先去辅助索引中查询数据，找到数据对应的 key 之后，再通过 key 回表到聚簇索引树查找数据）

2. mysql 事务特性

原子性：一个事务内的操作统一成功或失败

一致性：事务前后的数据总量不变

隔离性：事务与事务之间相互不影响

持久性：事务一旦提交发生的改变不可逆

3. 事务靠什么保证

原子性：由 undolog 日志保证，他记录了需要回滚的日志信息，回滚时撤销已执行的 sql

一致性：由其他三大特性共同保证，是事务的目的

隔离性：由 MVCC 保证

持久性：由 redolog 日志和内存保证，mysql 修改数据时内存和 redolog 会记录操作，宕机时可恢复

4. 事务的隔离级别

在高并发情况下，并发事务会产生脏读、不可重复读、幻读问题，这时需要用隔离级别来控制

读未提交： 允许一个事务读取另一个事务已提交的数据，可能出现不可重复读，幻读。

读提交： 只允许事务读取另一个事务没有提交的数据可能出现不可重复读，幻读。

可重复读： 确保同一字段多次读取结果一致，可能出现欢幻读。

可串行化： 所有事务逐次执行，没有并发问日

Inno DB 默认隔离级别为可重复读级别，分为快照度和当前读，并且通过间隙锁解决了幻读问题。

5. 什么是快照读和当前读

*快照读读取的是当前数据的可见版本，可能是会过期数据，不加锁的 select 就是快照都

*当前读读取的是数据的最新版本，并且当前读返回的记录都会上锁，保证其他事务不会并发修改这条记录。如 update、insert、delete、select for undate（排他锁）、select lockin share mode（共享锁） 都是当前读

6. MVCC 是什么

MVCC 是多版本并发控制，为每次事务生成一个新版本数据，每个事务都由自己的版本，从而不加锁就决绝读写冲突，这种读叫做快照读。只在读已提交和可重复读中生效。

实现原理由四个东西保证，他们是

undolog 日志：记录了数据历史版本

readView:事务进行快照读时动态生成产生的视图，记录了当前系统中活跃的事务 id，控制哪个历史版本对当前事务可见

隐藏字段 DB_TRC_ID： 最近修改记录的事务 ID

隐藏字段 DB_Roll_PTR： 回滚指针，配合 undolog 指向数据的上一个版本

7. MySQL 有哪些索引

主键索引：一张表只能有一个主键索引，主键索引列不能有空值和重复值

唯一索引：唯一索引不能有相同值，但允许为空

普通索引：允许出现重复值

组合索引：对多个字段建立一个联合索引，减少索引开销，遵循最左匹配原则

全文索引：myisam 引擎支持，通过建立倒排索引提升检索效率，广泛用于搜索引擎

8. 聚簇索引和非聚簇索引的区别

聚簇索引：聚簇索引的叶子节点存放的是主键值和数据行；辅助索引（在聚簇索引上创建的其它索引）的叶子节点存放的是主键值或指向数据行的指针。

优点：根据索引可以直接获取值，所以他获取数据更快；对于主键的排序查找和范围查找效率更高；

缺点：如果主键值很大的话，辅助索引也会变得很大；如果用 uuid 作为主键，数据存储会很稀疏；修改主键或乱序插入会让数据行移动导致页分裂；所以一般我们定义主键时尽量让主键值小，并且定义为自增和不可修改。

非聚簇索引（辅助索引）：叶子节点存放的是数据行地址，先根据索引找到数据地址，再根据地址去找数据

他们都是 b+数结构

9. MySQL 如何做慢 SQL 优化

可以查看执行计划分析数据的扫描类型、索引是否生效，常见的慢查询优化有：

（1）尽量减少 select 的数据列，尽量使用覆盖索引

（2）orderby 查找时使用索引进行排序，否则的话需要进行回表

（3）groupby 查询时，同样要用索引，避免使用到临时表

（4）分页查询时，如果 limit 后面的数字太大，可以使用子查询查出主键，再 limit 主键后 n 条数据就能走覆盖索引

（5） 使用复杂查询时，使用关联查询来代替子查询，并且最好使用内连接

（6）使用 count 函数时直接使用 count 的话 count(*)的效率最高

count(*)或 count(唯一索引)或 count(数字):表中总记录数，count(字段)不会统计 null

(7) 在写 update 语句时，where 条件要使用索引，否则会锁会从行锁升级为表锁

（8）表中数据是否太大，是不是要分库分表

10.为什么要用内连接而不用外连接？

用外连接的话连接顺序是固定死的，比如 left join，他必须先对左表进行全表扫描，然后一条条到右表去匹配；而内连接的话 mysql 会自己根据查询优化器去判断用哪个表做驱动。

子查询的话同样也会对驱动表进行全表扫描，所以尽量用小表做驱动表。

11.MySQL 整个查询的过程

（1）客户端向 MySQL 服务器发送一条查询请求
（2）服务器首先检查查询缓存，如果命中缓存，则返回存储在缓存中的结果。否则进入下一阶段
（3）服务器进行 SQL 解析、预处理、再由优化器生成对应的执行计划
（4）MySQL 根据执行计划，调用存储引擎的 API 来执行查询
（5）将结果返回给客户端，同时缓存查询结果
注意：只有在 8.0 之前才有查询缓存，8.0 之后查询缓存被去掉了

12.执行计划中有哪些字段？

我们想看一个 sql 的执行计划使用的语句是 explain+SQL，表中的字段包括：

type:扫描类型，效率从底到高为 ALL（全表扫描)>index(全索引扫描，我们的需要的数据在索引中可以获取)>range(使用索引进行范围查找)>ref(使用非唯一索引列进行了关联查询)>eq_ref (使用唯一索引进行关联查询)>const(使用唯一索引查询一行数据)>system(表中只有一行数据)

extra（额外的）:mysql 如何查询额外信息，常见的有：

filesort:在排序缓冲区中进行排序，需要回表查询数据

index:表示使用覆盖索引

index scan:排序时使用了索引排序，但如果是按照降序排序的话就会使用反向扫描索引

temporary:查询时要建立一个临时表存放数据

rows:找到了多少行数据

key:实际使用到的索引

id:select 查询的优先级，id 越大优先级越高，子查询的 id 一般会更大

select_type:查询的类型,是普通查询还是联合查询还是子查询，常见类型有 simple（不包含子查询），primary（标记复杂查询中最外层的查询），union(标记 primart 只后子查询)

table：者一行的数据是数哪张表的

possible_keys（可能的）:当前查询语句可能用到的索引，可能为 null(如果用了索引但是为 null 有可能是表数据太少 innodb 认为全表扫描更快)

ref（编号）:显示索引的哪一行被使用了

13.哪些情况索引会失效

（1）where 条件中有 or，除非所有查询条件都有索引，否则失效

（2）like 查询用%开头，索引失效

（3）索引列参与计算，索引失效

（4）违背最左匹配原则，索引失效

（5）索引字段发生类型转换，索引失效

（6）mysql 觉得全表扫描更快时（数据少），索引失效

14.B 和 B+数的区别，为什么使用 B+数

二叉树：索引字段有序，极端情况会变成链表形式

AVL 数：树的高度不可控

B 数：控制了树的高度，但是索引值和 data 都分布在每个具体的节点当中，若要进行范围查询，要进行多次回溯，IO 开销大

B+树：非叶子节点只存储索引值，叶子节点再存储索引+具体数据，从小到大用链表连接在一起，范围查询可直接遍历不需要回溯 7

15. MySQL 有哪些锁

基于粒度：

*表级锁：对整张表加锁，粒度大并发小

*行级锁：对行加锁，粒度小并发大

*间隙锁：间隙锁，锁住表的一个区间，间隙锁之间不会冲突只在可重复读下才生效，解决了幻读

基于属性：

*共享锁：又称读锁，一个事务为表加了读锁，其它事务只能加读锁，不能加写锁

*排他锁：又称写锁，一个事务加写锁之后，其他事务不能再加任何锁，避免脏读问题

16. Mysql 内连接、左连接、右连接的区别

内连接取量表交集部分，左连接取左表全部右表匹部分，右连接取右表全部坐表匹部分

17. sql 执行顺序

我单独写了一篇文章 http://t.csdn.cn/6a5Y3

18. 如何设计数据库？

（1）抽取实体，如用户信息，商品信息，评论

（2）分析其中属性，如用户信息：姓名、性别...

（3）分析表与表之间的关联关系

然后可以参考三大范式进行设计，设计主键时，主键要尽量小并且定义为自增和不可修改。

19.where 和 having 的区别？

where 是约束声明，having 是过滤声明，where 早于 having 执行，并且 where 不可以使用聚合函数，having 可以

20.三大范式

第一范式：每个列都不可以再拆分。

第二范式：在第一范式的基础上，非主键列完全依赖于主键，而不能是依赖于主键的一部分。

第三范式：在第二范式的基础上，非主键列只依赖于主键，不依赖于其他非主键。

21.char 和 varchar 的区别

char 是不可变的，最大长度为 255，varchar 是可变的字符串，最大长度为 2^16

22.InnoDB 什么情况下会产生死锁

事务 1 已经获取数据 A 的写锁，想要去获取数据 B 的写锁，然后事务 2 获取了 B 的写锁，想要去获取 A 的写锁，相互等待形成死锁。
mysql 解决死锁的机制有两个：1.等待， 直到超时 2.发起死锁检测，主动回滚一条事务
死锁检测的原理是构建一个以事务为顶点、锁为边的有向图，判断有向图是否存在环，存在即有死锁。
我们平时尽量减少事务操作的资源和隔离级别

23.MySQL 删除自增 id，随后重启 MySQL 服务，再插入数据，自增 id 会从几开始？

innodb 引擎：

MySQL8.0 前，下次自增会取表中最大 id + 1。原理是最大 id 会记录在内存中，重启之后会重新读取表中最大的 id
MySQL8.0 后，仍从删除数据 id 后算起。原理是它将最大 id 记录在 redolog 里了

myisam：
自增的 id 都从删除数据 id 后算起。原理是它将最大 id 记录到数据文件里了

24.MySQL 插入百万级的数据如何优化？

（1）一次 sql 插入多条数据，可以减少写 redolog 日志和 binlog 日志的 io 次数（sql 是有长度限制的，但可以调整）

（2）保证数据按照索引进行有序插入

（3）可以分表后多线程插入

五.常用开发框架系列

1. 什么是 Spring？

Spring 是个轻量级的框架，通过 IOC 达到松耦合的目的，通过 AOP 可以分离应用业务逻辑和系统服务进行内聚性的开发，不过配置各种组件时比较繁琐，所以后面才出选了 SpringBoot 的框架。

2. IOC 是什么？

IOC 是控制反转，是一种思想，把对象的创建和调用从程序员手中交由 IOC 容器管理，降低对象之间的依赖关系。

创建一个 bean 的方式有 xml 方式、@Bean 注解方式、@Componte 方式

我们在对一个 bean 进行实例化后，要对他的属性进行填充，大多数我们都是使用 @Autowire 直接的填充依赖注入的，他是有限按照类型进行匹配。

3. AOP 是什么？

AOP 是面向切面编程，可以将那些与业务不相关但是很多业务都要调用的代码抽取出来，思想就是不侵入原有代码的情况下对功能进行增强。

SpringAOP 是基于动态代理实现的，动态代理是有两种，一种是 jdk 动态代理，一种是 cglib 动态代理;

jdk 动态代理是原理是利用反射来实现的，需要调用反射包下的 Proxy 类的 newProxyInstance 方法来返回代理对象，这个方法中有三个参数，分别是用于加载代理类的类加载器，被代理类实现的接口的 class 数组和一个用于增强方法的 InvocaHandler 实现类。

cglib 动态代理原理是利用 asm 开源包来实现的，是把被代理类的 class 文件加载进来，通过修改它的字节码生成子类来处理

jdk 动态代理要求被代理类必须有实现的接口，生成的动态代理类会和代理类实现同样的接口，cglib 则，生成的动态代理类会继承被代理类。Spring 默认使用 jdk 动态代理，当被代理的类没有接口时就使用 cglib 动态代理

4. 如何定义一个全局异常处理类？

想要定义一个全局异常处理类的话，我们需要在这个类上添加@ContaollerAdvice 注解，然后定义一些用于捕捉不同异常类型的方法，在这些方法上添加@ExceptionHandler(value = 异常类型.class)和@ResponseBody 注解，方法参数是 HttpServletRequest 和异常类型，然后将异常消息进行处理。

如果我们需要自定义异常的话，就写一个自定义异常类，该类需要继承一个异常接口，类属性包括 final 类型的连续 id、错误码、错误信息，再根据需求写构造方法；

5. 如何使用 aop 自定义日志？

第一步：创建一个切面类，把它添加到 ioc 容器中并添加@Aspect 注解

第二步： 在切面类中写一个通知方法，在方法上添加通知注解并通过切入点表达式来表示要对哪些方法进行日志打印，然后方法参数为 JoinPoint

第三步：通过 JoinPoint 这个参数可以获取当前执行的方法名、方法参数等信息，这样就可以根据需求在方法进入或结束时打印日志

6. 循环依赖是什么，怎么解决的？

循环依赖就是在创建 A 实例的时候里面包含着 B 属性实例，所以这个时候就需要去创建 B 实例，而创 建 B 实例过程中也包含着 A 实例。 这样 A 实例还在创建的过程当中，所以就导致 A 和 B 实例都创建不出来。

spring 通过三级缓存来解决循环依赖：

一级缓存：单例池，缓存经过了已经初始化完毕的 Bean

二级缓存 ：半成品池，缓存还未初始化完毕的 Bean

三级缓存：缓存的是获取 Bean 的代理对象的表达式

我们在创建 A 的过程中，先将 A 放入三级缓存 ，这时要创建 B，B 要创建 A 就直接去三级缓存中查找，并且判断需不需要进行 AOP 处理，如果需要就在三级缓存中获取 A 的代理对象，不需要就取 A 原始对象。然后将取出的对象放入二级缓存中，这个时候其他需要依赖 A 对象的直接从二级缓存中去获取即可。当 B 初始化完成进入一级缓存后，A 继续执行生命周期，当 A 完成了属性的注入后，就可以放入一级缓存了

spring2.6 之前默认会解决循环依赖。在 spring2.6 之后需要通过配置开启解决循环依赖

7. Bean 的作用域

（1）Singleton:一个 IOC 容器只有一个

（2）Prototype:每次调用 getBean()都会生成一个新的对象

（3）request:每个 http 请求都会创建一个自己的 bean

（4）session:同一个 session 共享一个实例

（5）application:整个 serverContext 只有一个 bean

（6）webSocket:一个 websocket 只有一个 bean

8. Bean 生命周期

实例化 Instantiation->属性赋值 Populate->初始化 Initialization->销毁 Destruction
在这四步的基础上面，Spring 提供了一些拓展点：

*Bean 自身的方法: 包括了 Bean 本身调用的方法和通过配置文件中的 init-method 和 destroy-method 指定的方法
*Bean 级生命周期接口方法:包括了 BeanNameAware、BeanFactoryAware、InitializingBean 和 DiposableBean 这些接口的方法
*容器级生命周期接口方法：包括了 InstantiationAwareBeanPostProcessor 和 BeanPostProcessor 这两个接口实现，一般称它们的实现类为“后处理器”。
*工厂后处理器接口方法: 包括了 AspectJWeavingEnabler, ConfigurationClassPostProcessor, CustomAutowireConfigurer 等等非常有用的工厂后处理器接口的方法。工厂后处理器也是容器级的。在应用上下文装配配置文件之后立即调用。

9. Spring 事务原理？

spring 事务有编程式和声明式，我们一般使用声明式，在某个方法上增加@Transactional 注解，这个方法中的 sql 会统一成功或失败。

原理是：

当一个方法加上@Transactional 注解，spring 会基于这个类生成一个代理对象并将这个代理对象作为 bean，当使用这个 bean 中的方法时，如果存在@Transactional 注解，就会将事务自动提交设为 false，然后执行方法，执行过程没有异常则提交，有异常则回滚、

10. spring 事务失效场景

（1）事务方法所在的类没有加载到容器中

（2）事务方法不是 public 类型

（3）同一类中，一个没有添加事务的方法调用另外以一个添加事务的方法，事务不生效

（4）spring 事务默认只回滚运行时异常，可以用 rollbackfor 属性设置

（5）业务自己捕获了异常，事务会认为程序正常秩序

11. spring 事务的隔离级别

default:默认级别，使用数据库自定义的隔离级别

其它四种隔离级别与 mysql 一样

12. spring 事务的传播行为

（1）支持当前事务，如果不存在，则新启一个事务

（2）支持当前事务，如果不存在，则抛出异常

（3）支持当前事务，如果不存在，则以非事务方式执行

（4）不支持当前事务，创建一个新事物

（5）不支持当前事务，如果已存在事务就抛异常

（6）不支持当前事务，始终以非事务方式执行

13. spring 用了哪些设计模式

BeanFactory 用了工厂模式，AOP 用了动态代理模式，RestTemplate 用来模板方法模式，SpringMVC 中 handlerAdaper 用来适配器模式，Spring 里的监听器用了观察者模式

14. SpringMV 工作原理

SpringMVC 工作过程围绕着前端控制器 DispatchServerlet，几个重要组件有 HandleMapping（处理器映射器）、HandleAdapter（处理器适配器）、ViewReslover（试图解析器）

工作流程：

（1）DispatchServerlet 接收用户请求将请求发送给 HandleMapping

（2）HandleMapping 根据请求 url 找到具体的 handle 和拦截器，返回给 DispatchServerlet

（3）DispatchServerlet 调用 HandleAdapter,HandleAdapter 执行具体的 controller，并将 controller 返回的 ModelAndView 返回给 DispatchServler

（4）DispatchServerlet 将 ModelAndView 传给 ViewReslover,ViewReslover 解析后返回具体 view

（5）DispatchServerlet 根据 view 进行视图渲染，返回给用户

15. springboot 自动配置原理

在 spring—boot—autoconfigura 包下存放了 spring 内置的自动配置类和 spring.factories 文件，这个文件中存放了这些配置类的全类名 ；

启动类@SpringbootApplication 注解下，有三个关键注解

（1）@springbootConfiguration:表示启动类是一个自动配置类

（2）@CompontScan:扫描启动类所在包下及子包的组件到容器中

（3）@EnableConfigutarion，下面有个子注解@Import 会导入上面所说的自动配置类，这些配置类会根据元注解的装配条件生效，生效的类就会被实例化，加载到 ioc 容器中；这些自动配置类还会通过 xxxProperties 文件里配置来进行属性设值

16 .springboot 常用注解

@RestController ：修饰类，该控制器会返回 Json 数据

@RequestMapping("/path") ：修饰类，该控制器的请求路径

@Autowired ： 修饰属性，按照类型进行依赖注入

@PathVariable ： 修饰参数，将路径值映射到参数上

@ResponseBody :修饰方法，该方法会返回 Json 数据

@RequestBody（需要使用 Post 提交方式） :修饰参数，将 Json 数据封装到对应参数中
@Controller@Service@Compont: 将类注册到 ioc 容器

@Transaction：开启事务

16. spring 的 bean 是线程安全的吗？

spring 的默认 bean 作用域是单例的，单例的 bean 不是线程安全的，但是开发中大部分的 bean 都是无状态的，不具备存储功能，比如 controller、service、dao，他们不需要保证线程安全。

如果要保证线程安全，可以将 bean 的作用域改为 prototype，比如像 Model View。

另外还可以采用 ThreadLocal 来解决线程安全问题。ThreadLocal 为每个线程保存一个副本变量，每个线程只操作自己的副本变量。

17. springcloud 主要解决什么问题？

解决服务之间的通信、容灾、负载平衡、冗余问题，能方便服务集中管理，常用组件有注册中心、配置中心、远程调用。服务熔断、网关

18. CAP 理论

C：一致性，这里指的强一致性，也就是数据更新完，访问任何节点看到的数据完全一致

A：可用性，就是任何没有发生故障的服务必须在规定时间内返回合正确结果

P：容灾性，当网络不稳定时节点之间无法通信，造成分区，这时要保证系统可以继续正常服务。提高容灾性的办法就是把数据分配到每一个节点当中，所以 P 是分布式系统必须实现的，然后需要在 C 和 A 中取舍

19. 为什么不能同时保证一致性和可用性呢？

当网络发生故障时，如果要保障数据一致性，那么节点相互间就只能阻塞等待数据真正同步时再返回，就违背可用性了。如果要保证可用性，节点要在有限时间内将结果返回，无法等待其它节点的更新消息，此时返回的数据可能就不是最新数据，就违背了一致性了

20.熔断限流的理解？

SprngCloud 中用 Hystrix 组件来进行降级、熔断、限流

熔断是对于消费者来讲，当对提供者请求时间过久时为了不影响性能就对链接进行熔断，

限流是对于提供者来讲，为了防止某个消费者流量太大，导致其它更重要的消费者请求无法及时处理。限流可用通过拒绝服务、服务降级、消息队列延时处理、限流算法来实现

21.常用限流算法

计数器算法：使用 redis 的 setnx 和过期机制实现

漏桶算法：一般使用消息队列来实现，系统以恒定速度处理队列中的请求，当队列满的时候开始拒绝请求。

令牌桶算法：计数器算法和漏桶算法都无法解决突然的大并发，令牌桶算法是预先往桶中放入一定数量 token，然后用恒定速度放入 token 直到桶满为止，所有请求都必须拿到 token 才能访问系统

六.Redis 系列

1. redis 为什么快？

（1）完全基于内存操作

（2）数据结构简单，对数据操作简单

（3）redis 执行命令是单线程的，避免了上下文切换带来的性能问题，也不用考虑锁的问题

(4) 采用了非阻塞的 io 多路复用机制，使用了单线程来处理并发的连接;内部采用的 epoll+自己实现的事件分离器

其实 Redis 不是完全多线程的，在核心的网络模型中是多线程的用来处理并发连接，但是数

据的操作都是单线程。Redis 坚持单线程是因为 Redis 是的性能瓶颈是网络延迟而不是 CPU，多线程对数据读取不会带来性能提升。

2. redis 持久化机制

（1）快照持久化 RDB

redis 的默认持久化机制，通过父进程 fork 一个子进程，子进程将 redis 的数据快照写入一个临时文件，等待持久化完毕后替换上一次的 rdb 文件。整个过程主进程不进行任何的 io 操作。持久化策略可以通过 save 配置单位时间内执行多少次操作触发持久化。所以 RDB 的优点是保证 redis 性能最大化，恢复速度数据较快，缺点是可能会丢失两次持久化之间的数据

（2）追加持久化 AOF

以日志形式记录每一次的写入和删除操作，策略有每秒同步、每次操作同步、不同步，优点是数据完整性高，缺点是运行效率低，恢复时间长

3. Redis 如何实现 key 的过期删除？

采用的定期过期+惰性过期

定期删除 ：Redis 每隔一段时间从设置过期时间的 key 集合中，随机抽取一些 key ，检查是否过期，如果已经过期做删除处理。
惰性删除 ：Redis 在 key 被访问的时候检查 key 是否过期，如果过期则删除。

4. Redis 数据类型应用场景

String：可以用来缓存 json 信息，可以用 incr 命令实现自增或自减的计数器

Hash：与 String 一样可以保存 json 信息

List：可以用来做消息队列，list 的 pop 是原子性操作能一定程度保证线程安全

Set：可以做去重，比如一个用户只能参加一次活动 ;可以做交集求共友

SortSet ：有序的。可以实现排行榜

5. Redis 缓存穿透如何解决？

缓存穿透是指频繁请求客户端和缓存中都不存在的数据，缓存永远不生效，请求都到达了数据库。

解决方案：

（1）在接口上做基础校验，比如 id<=0 就拦截

（2）缓存空对象：找不到的数据也缓存起来，并设置过期时间，可能会造成短期不一致

（3）布隆过滤器：在客户端和缓存之间添加一个过滤器，拦截掉一定不存在的数据请求

6. Redis 如何解决缓存击穿？

缓存击穿是值一个 key 非常热点，key 在某一瞬间失效，导致大量请求到达数据库

解决方案：

（1）设置热点数据永不过期

（2）给缓存重建的业务加上互斥锁，缺点是性能低

7. Redis 如何解决缓存雪崩？

缓存雪崩是值某一时间 Key 同时失效或 redis 宕机，导致大量请求到达数据库

解决方案：

（1）搭建集群保证高可用

（2）进行数据预热，给不同的 key 设置随机的过期时间

（3）给缓存业务添加限流降级，通过加锁或队列控制操作 redis 的线程数量

（4）给业务添加多级缓存

8. Redis 分布式锁的实现原理

原理是使用 setnx+setex 命令来实现，但是会有一系列问题：

（1）任务时常超过缓存时间，锁自动释放。可以使用 Redision 看门狗解决

（2）加锁和释放锁的不是同一线程。可以在 Value 中存入 uuid，删除时进行验证。但是要注意验证锁和删除锁也不是一个原子性操作，可以用 lua 脚本使之成为原子性操作

（3）不可重入。可以使用 Redision 解决（实现机制类似 AQS,计数）

（4）redis 集群下主节点宕机导致锁丢失。使用红锁解决

9. Redis 集群方案

(1)主从模式：个 master 节点，多个 slave 节点，master 节点宕机 slave 自动变成主节点

(2)哨兵模式：在主从集群基础上添加哨兵节点或哨兵集群，用于监控 master 节点健康状态，通过投票机制选择 slave 成为主节点

(3)分片集群：主从模式和哨兵模式解决了并发读的问题，但没有解决并发写的问题，因此有了分片集群。分片集群有多个 master 节点并且不同 master 保存不同的数据，master 之间通过 ping 相互监测健康状态。客户端请求任意一个节点都会转发到正确节点，因为每个 master 都被映射到 0-16384 个插槽上，集群的 key 是根据 key 的 hash 值与插槽绑定

10. Redis 集群主从同步原理

主从同步第一次是全量同步：slave 第一次请求 master 节点会根据 replid 判断是否是第一次同步，是的话 master 会生成 RDB 发送给 slave。

后续为增量同步：在发送 RDB 期间，会产生一个缓存区间记录发送 RDB 期间产生的新的命令,slave 节点在加载完后，会持续读取缓存区间中的数据

11. Redis 缓存一致性解决方案

Redis 缓存一致性解决方案主要思考的是删除缓存和更新数据库的先后顺序

先删除缓存后更新数据库存在的问题是可能会数据不一致，一般使用延时双删来解决，即先删除缓存，再更新数据库，休眠 X 秒后再次淘汰缓存。第二次删除可能导致吞吐率降低，可以考虑进行异步删除。

先更新数据库后删除缓存存在的问题是会可能会更新失败，可以采用延时删除。但由于读比写快，发生这一情况概率较小。

但是无论哪种策略，都可能存在删除失败的问题，解决方案是用中间件 canal 订阅 binlog 日志提取需要删除的 key，然后另写一段非业务代码去获取 key 并尝试删除，若删除失败就把删除失败的 key 发送到消息队列，然后进行删除重试。

12. Redis 内存淘汰策略

当内存不足时按设定好的策略进行淘汰，策略有(1)淘汰最久没使用的（2）淘汰一段时间内最少使用的（3）淘汰快要过期的

八、场景题

1. Java 如何实现统计在线人数的功能？

博主写了另外一篇文章：http://t.csdn.cn/Q1S4h

2. 电商网站可以分成哪些模块（或订单模块要完成哪些功能）？

用户模块（用户账户、会员等级、收货信息）、订单模块（订单编号、类型信息、状态信息、时间信息等）、商品模块（店铺信息、数量、价格等）、支付模块（支付方式、支付时间、支付单号等）、物流模块（物流公司、物流单号、物流状态等）

九.其他（RabitMQ、数据结构与算法、nginx、git、jwt 登录等...）

1. RabbitMQ 如何保证消息不丢失？

发送端：

（1）创建一个消息状态表，开启 RabbitMQ 的 confirm 机制，当收到 MQ 回传的 ACK 以更新消息状态

（2）开启定时任务，隔断时间重新发送状态表中超时的任务，多次投递失败进行报警。

消费端：

（1）为了避免消息重复消费，要增加一张消息处理表，消费者拿到消息时判断消息处理表中当前消息是否已经存在，已经存在就抛弃，不存在就进行消费并放入记录表，消费和放入记录表要放在一个事务当中。

（2)开启手动 ack 模式，在消费者处理完业务后才返回 ACK，避免消息还没有处理完就 ACK。

具体可以看博主另外一篇文章：(152 条消息) RabbitMQ 消息丢失的场景，如何保证消息不丢失？（详细讲解，一文看懂）_十八岁讨厌 Java 的博客-CSDN 博客

2. RabbitMQ 如何保证消费顺序

RabbitMQ 消费顺序乱了是因为消费者集群拿到消息后对消息处理速度不同导致的，比如可能将增删改变成了增改删。

解决方法：为 RabbitMQ 创建多个 Queue,每个消费者只监听其中一个 Queue,同一类型的消息都放在一个 queue 中，同一个 queue 的消息是一定会保证有序的。

3. 设计模式六大原则

（1）单一职责原则：一个类或者一个方法只负责一项职责，尽量做到类只有一个行为引起变化；
（2）里氏替换原则：子类可以扩展父类的功能，但不能改变原有父类的功能

（3）依赖倒置原则：高层模块不应该依赖底层模块，两者都应该依赖接口或抽象类
（4）接口隔离原则：建立单一接口，尽量细化接口
（5）迪米特原则：只关心其它对象能提供哪些方法，不关心过多内部细节
（6）开闭原则：对于拓展是开放，对于修改是封闭的

4. 设计模式分类

创建型模式：主要是描述对象的创建，代表有单例、原型模式、工厂方法、抽象工厂、建造者模式

结构型模式：主要描述如何将类或对象按某种布局构成更大的结构，代表有代理、适配器、装饰

行为型模式：描述类或对象之间如何相互协作共同完成单个对象无法完成的任务，代表有模板方法模式、策略模式、观察者模式、备忘录模式

5. 排序算法的时间复杂度

交换排序：冒泡排序（n^2,稳定）,快速排序(nlogn，不稳定)
选择排序：直接选择排序（n^2,不稳定），堆排序(nlogn,不稳定 s)，
插入排序：直接插入排序（n^2,稳定），希尔排序(N^1.25，不稳定)
归并排序(nlogn，稳定)

6. 大量数据排名，采用什么数据结构

当数据很大时，并且有序程度低时，堆排序最快;当数据很大时，并且有序程度高时，快速排序最快

7. 二叉树和堆之间联系或区别

堆是一种特殊的二叉树，所有父结点都比子结点要小的完全二叉树我们称为最小堆，所有父结点都比子结点要大，这样的完全二叉树称为最大堆。

8. 平衡二叉树不平衡如何调整？

按照不平衡的情况有四种调整方法，分别是 LR、RL、LL、RR 调整

当不平衡的子树以当前节点为第一个节点往下再数到第三个节点的路径是先左再右时使用 LR 调整，LR 是先将第二个节点旋转到第三个节点的左边，将第一个节点移动到第三个节点的右边；

RL 与 LR 相反，RR 是把第一个节点移到第三个节点左边，RR 与 LL 相反

9. hash 表冲突的解决方法

开放地址法：有线性探测法和平方探测法，当发生冲突时，继续往后找
再哈希法：构造多个哈希函数，发生冲突后使用下一个函数
链地址法：将 hash 值相同的记录用链表链接起来
建立公共溢出区：将哈希表分为基础表和益处表两部分，发生冲突的填入益处表

10. cookie 和 session 的联系

因为 http 协议是无状态的，无法识别两次请求是否来自同一个客户端，于是就有了 cookie 和 session 的概念。

cookies:是存放在客户浏览器中，每次 http 请求都会携带，可以用来告知服务端两个请求是否来自同一浏览器，cookied 的单个数据大小和存储个数是有限制的，不同浏览器限制不同，cookie 的安全性较低。

session：当客户端第一次请求服务器时服务器为这个请求分配的一块区域，的存储结构为 ConcurrentHashMap，服务器会将 sessionId 返回给客户端并存入 cookie。相比 cookie 他的安全性更高，但失效时间较短

接下来客户端每次请求带着 cookie，服务端会从中获取 sessionId 来进行匹配，用这套机制就实现了服务器和客户端进行有记忆的对话。

11. Nginx 反向代理是什么，负载均衡算法有哪些？

反向代理是用来代理服务器接收请求的，然后将请求转发给内部网络的服务器，并将从服务

器上得到的结果返回给客户端，此时代理服务器对外就表现为一个服务器。

负载均衡算法有：轮询（默认）、带权轮询、ip_hash（按 ip 哈希结果分配，能解决 session 共享问题）、url_hash(按访问的 URL 的哈希结果分配)、fair（根据服务端响应时间分配，响应时间短优先）

九、秒杀项目相关问题：

1. 项目流程

用户点击下单按钮时，进行三次判断：先判断请求路径是否合法,因为做了动态 URL；再判断用户是否已经下单过，就是看 redis 缓存中有没有用户下单信息；最后判断库存，这里进行了 redis 库存预减，由于判断库存和预减库存不是原子性操作，所以用 lua 脚本来执行这一段代码。然后从这里开始使用分布式锁，锁 id 为用户 id+商品 id,防止一个用户发送多次请求让 redis 多次预减。

Redis 扣减成功后，进行异步下单，直接将正在处理返回给前端，将用户 id 和商品 Id 发送 RabbitMQ 中，负责下单的业务会从消息队列中拿出消息，去执行以下操作：

1.减库存，减库存时用 where 库存>0 防止超卖

2.订单表中生成记录，订单表中的用户 id 和商品 id 添加了联合唯一索引防止超卖

减库存和增加订单放在一个事务内保证一致性

3.将用户 id 和订单 id 缓存到 redis 中用来最初对用户重复下单的判断

4.释放分布式锁，根据 value 去判断锁是不是当前线程的，判断和删除锁不是原子性操作，所以封装到了 lua 脚本中

2. 提升 qps 的操作

（1）页面动静分离，静态页面缓存到 redis
（2）分布式锁拦截不同用户的重复
（3）限流算法
（4）验证码限流
（5）rabbitMq 流量削峰

（6）接口隐藏

微学院相关问题
1.如何用 springSecurity 做的认证授权？
在数据库中有五张表，分别是菜单表，角色表，用户表，他们是多对多的关系，所以还有角色菜单表，角色用户表

登录后进入认证过滤器，获取用户名和密码，根据用户名查询用户具有的权限并把用户名和对应权限信息放到 redis，JWT 生成 token 后放入 cookie,每次调用接口时携带
然后执行授权过滤器,从 header 中获取 token 解析出用户名，根据用户名从 redis 中获取权限列表，然后 springSecurity 就能够判断当前请求是否有权限访问

2.前后端联调经常遇到的问题：
1.请求方式不匹配

2.json、x-wwww-form-urlencoded 混乱的错误

3.前后端参数不一致，空指针异常，数据类型不匹配

4.mp 生成的分布式 id 是 19 位，JavsScrip 只会处理 16 位，将 id 生成策略改为 String 类型

5.跨域问题：跨域问题是在访问协议、ip 地址、端口号这三个有任何一个不一样，相互访问就会出现跨域，可以通过 Spring 注解解决跨域的 @CrossOrigin，也可以使用 nginx 反向代理、网关

6.maven 加载项目时，默认不会加载 src-java 文件夹的 xml 类型文件，可以 将 xml 放到 resources 文件夹下，也可以在 yaml 和 pom 中添加配置