**尚硅谷Java技术之上海高频面试题**

**版本：V1.0**

**尚硅谷Java辅导部**

# 面试技巧篇

## 项目组人数

面试之前想好具体人数就行,类似于大家在硅谷学习过程中,一个小组有多少人一样

Eg 12人 其中 5人java 2人前端 3人测试 2人运维 (仅供参考)

## 离职原因

年底:客观原因 也即外界因素导致的 不可抗拒 的因素

其他时间:避免说老东家的不好,合理即可

## 为什么来上海

合理即可

Eg:有同学/朋友/家人 在上海 等 重要的不是问什么,而是当遇到此类问题自己是否能够自如应对

## 自己的优缺点

勇于说出的自己的优点,谈到缺点 稍微注意 尽量用自己的缺点衬托有点

## 有什么想问我的吗?

如果是技术面试官

Eg:项目进展 项目规模 项目选用技术 项目团队

如果是人事面试官

Eg:福利待遇等事关切身利益的问题

# Java基础篇

## 面向对象是什么,并解释

面向对象是利于语言对现实事物进行抽象。面向对象具有以下四大特征：

（1）继承：继承是从已有类得到继承信息创建新类的过程

（2）封装：通常认为封装是把数据和操作数据的方法绑定起来，对数据的访问只能通过已定义的接口。

（3）多态性：多态性是指允许不同子类型的对象对同一消息作出不同的响应。

（4）抽象：抽象是将一类对象的共同特征总结出来构造类的过程，包括数据抽象和行为抽象两方面。

## Object类中常见方法

（1）protected Object clone()--->创建并返回此对象的一个副本。

（2）boolean equals(Object obj)--->指示某个其他对象是否与此对象“相等”。

（3）protected void finalize()--->当垃圾回收器确定不存在对该对象的更多引用时，由对象的垃圾回收器调用此方法。

（4）Class<? extendsObject> getClass()--->返回一个对象的运行时类。

（5）int hashCode()--->返回该对象的哈希码值。

（6）void notify()--->唤醒在此对象监视器上等待的单个线程。

（7）void notifyAll()--->唤醒在此对象监视器上等待的所有线程。

（8）String toString()--->返回该对象的字符串表示。

（9）void wait()--->导致当前的线程等待，直到其他线程调用此对象的notify()方法或notifyAll()方法。

void wait(long timeout)--->导致当前的线程等待，直到其他线程调用此对象的notify()方法或notifyAll()方法，或者超过指定的时间量。

void wait(long timeout, int nanos)--->导致当前的线程等待，直到其他线程调用此对象的notify()

## 重载与重写有什么区别?

1、重载发生在本类，重写发生在父类与子类之间；

2、重载的方法名必须相同，重写的方法名相同且返回值类型必须相同；

3、重载的参数列表不同，重写的参数列表必须相同。

4、重写的访问权限不能比父类中被重写的方法的访问权限更低。

5、构造方法不能被重写

## 接口与抽象类的区别

抽象类要被子类继承，接口要被类实现。

接口可多继承接口，但类只能单继承。

抽象类可以有构造器、接口不能有构造器

抽象类：除了不能实例化抽象类之外，它和普通Java类没有任何区别

抽象类：抽象方法可以有public、protected和default这些修饰符、接口：只能是public

抽象类：可以有成员变量；接口：只能声明常量

## Stringbuffer Stringbuilder String区别

（1）StringBuffer 与StringBuilder中的方法和功能完全是等价的，

（2）只是StringBuffer 中的方法大都采用了 synchronized 关键字进行修饰，因此是线程安全的，而StringBuilder没有这个修饰，可以被认为是线程不安全的。

（3）在单线程程序下，StringBuilder效率更快，因为它不需要加锁，不具备多线程安全而StringBuffer则每次都需要判断锁，效率相对更低

（4）缓冲区

StringBuffer 每次获取 toString 都会直接使用缓存区的 toStringCache 值来构造一个字符串。

而 StringBuilder 则每次都需要复制一次字符数组，再构造一个字符串。

所以，缓存冲这也是对 StringBuffer 的一个优化吧，不过 StringBuffer 的这个toString 方法仍然是同步的。

StringBuffer 代码片段：

private transient char[] toStringCache; @Override public synchronized String toString() {     if (toStringCache == null) {         toStringCache = Arrays.copyOfRange(value, 0, count);     }     return new String(toStringCache, true); }

StringBuilder 代码片段：

@Override public String toString() {     // Create a copy, don't share the array     return new String(value, 0, count); }

## ArrayList和LinkedList有什么区别？

（1）ArrayList是实现了基于动态数组的数据结构，LinkedList基于链表的数据结构。

（2）对于随机访问get和set，ArrayList觉得优于LinkedList，因为LinkedList要移动指针。

（3）对于新增和删除操作add和remove，LinkedList比较占优势，因为ArrayList要移动数据。 这一点要看实际情况的。若只对单条数据插入或删除，ArrayList的速度反而优于LinkedList。但若是批量随机的插入删除数据，LinkedList的速度大大优于ArrayList.因为ArrayList每插入一条数据，要移动插入点及之后的所有数据。

## sleep和wait在线程里有什么区别？

sleep方法：

属于Thread类中的方法；会导致程序暂停执行指定的时间，让出cpu该其他线程，但是他的监控状态依然保持着，当指定时间到了之后，又会自动恢复运行状态；在调用sleep方法的过程中，线程不会释放对象锁。（只会让出CPU，不会导致锁行为的改变）

wait方法：

属于Object类中的方法；在调用wait方法的时候，线程会放弃对象锁，进入等待此对象的等待锁定池，只有针对此对象调用notify方法后本线程才进入对象锁定池准备。获取对象锁进入运行状态。（不仅让出CPU，还释放已经占有的同步资源锁

## final、finally、finalize

final

在java中，final可以用来修饰类，方法和变量（成员变量或局部变量）

若父类中final方法的访问权限为private，将导致子类中不能直接继承该方法，因此，此时可以在子类中定义相同方法名的函数，此时不会与重写final的矛盾，而是在子类中重新地定义了新方法。

finally

finally 语句块还是没有执行，为什么呢？因为我们在 try 语句块中执行了 System.exit (0) 语句，终止了 Java 虚拟机的运行。那有人说了，在一般的 Java 应用中基本上是不会调用这个 System.exit(0) 方法的。OK ！没有问题，我们不调用 System.exit(0) 这个方法，那么 finally 语句块就一定会执行吗？

再一次让大家失望了，答案还是否定的。当一个线程在执行 try 语句块或者 catch 语句块时被打断（interrupted）或者被终止（killed），与其相对应的 finally 语句块可能不会执行。还有更极端的情况，就是在线程运行 try 语句块或者 catch 语句块时，突然死机或者断电，finally 语句块肯定不会执行了。可能有人认为死机、断电这些理由有些强词夺理，没有关系，我们只是为了说明这个问题。

finalize

finalize()是在java.lang.Object里定义的。这个方法在gc启动，该对象被回收的时候被调用。

## hashMap和hashTable区别？

1、两者父类不同

2、提供的接口不同，HashTable 多提供 elments() 和 contains()

elments()：用于返回Hashtable中的value的枚举

contains()：判断 是否包含传入的value

3、对null 的支持不同

Hashtable：kv都不能为 null

HashMap：k可以为null，但只能有一个（因为key的唯一性），v可以为null，且不唯一

4、安全性不同

5、初始容量大小和每次扩容大小不同

hashMap：初始大小：16，每次扩容2

hashTable：初始大小：11

6、计算hash值方法不同

## 反射原理

Java 反射机制是在运行状态中，对于任意一个类，都能够获得这个类的所有属性和方法，对于任意一个对象都能够调用它的任意一个属性和方法。这种在运行时动态的获取信息以及动态调用对象的方法的功能称为Java的反射机制。

Class 类与java.lang.reflect类库一起对反射的概念进行了支持，该类库包含了Field,Method,Constructor类(每个类都实现了Member接口)。这些类型的对象时由JVM在运行时创建的，用以表示未知类里对应的成员。

这样就可以使用Constructor 创建新的对象，用get()和set()方法读取和修改与Field对象关联的字段，用invoke()方法调用与Method对象关联的方法。另外，还可以调用getFields() getMethods()和getConstructors()等很便利的方法，以返回表示字段，方法，以及构造器的对象的数组。这样匿名对象的信息就能在运行时被完全确定下来，而在编译时不需要知道任何事情

## HashMap底层原理

HashMap基于哈希表的Map接口实现，是以key-value存储形式存在，即主要用来存放键值对。HashMap 的实现不是同步的，这意味着它不是线程安全的。它的key、value都可以为null。此外，HashMap中的映射不是有序的。

JDK1.8 之前 HashMap 由 数组+链表 组成的，数组是 HashMap 的主体，链表则是主要为了解决哈希冲突(两个对象调用的hashCode方法计算的哈希码值一致导致计算的数组索引值相同)而存在的（“拉链法”解决冲突）.JDK1.8 以后在解决哈希冲突时有了较大的变化，当链表长度大于阈值（或者红黑树的边界值，默认为 8）并且当前数组的长度大于64时，此时此索引位置上的所有数据改为使用红黑树存储。

补充：将链表转换成红黑树前会判断，即使阈值大于8，但是数组长度小于64，此时并不会将链表变为红黑树。而是选择进行数组扩容。

这样做的目的是因为数组比较小，尽量避开红黑树结构，这种情况下变为红黑树结构，反而会降低效率，因为红黑树需要进行左旋，右旋，变色这些操作来保持平衡 。同时数组长度小于64时，搜索时间相对要快些。所以综上所述为了提高性能和减少搜索时间，底层在阈值大于8并且数组长度大于64时，链表才转换为红黑树。具体可以参考 treeifyBin方法。

当然虽然增了红黑树作为底层数据结构，结构变得复杂了，但是阈值大于8并且数组长度大于64时，链表转换为红黑树时，效率也变的更高效。

## 说一下有哪些IO类

InputStream/Reader：所有的输入流的基类，前者是字节输入流，后者是字符输入流。

OutputStream/Writer：所有输出流的基类，前者是字节输出流，后者是字符输出流。

按照流的流向分，可以分为输入流和输出流；

按照操作单元划分，可以划分为字节流和字符流；

按照流的角色划分为节点流和处理流。

Java Io流共涉及40多个类，这些类看上去很杂乱，但实际上很有规则，而且彼此之间存在非常紧密的联系，Java I0流的40多个类都是从如下4个抽象类基类中派生出来的。

InputStream/Reader: 所有的输入流的基类，前者是字节输入流，后者是字符输入流。

OutputStream/Writer: 所有输出流的基类，前者是字节输出流，后者是字符输出流。

## 如果查看死锁

1.可以通过jstack命令来进行查看，jstack命令中会显示发生了死锁的线程

2,或者两个线程去操作数据库时，数据库发生了死锁，这是可以查询数据库的死锁情况

SOL

1、查询是否锁表

show OPEN TABLES wtere In\_use 》e;

2、查询进程

show processlist;

3、查看正在锁的事务

SELECT \* EROM INFORMATION\_SCHEMA.INNODB\_LOCKS;

4、查看等待锁的事务

SELECT \* FROM INFORMATION\_SCHEMA.INNODB\_LOCK\_WAITS;

## Java死锁如何避免

造成死锁的几个原因:

1.一个资源每次只能被一个线程使用

2.一个线程在阻塞等待某个资源时，不释放已占有资源

3.一个线程已经获得的资源，在未使用完之前，不能被强行剥夺

4.若干线程形成头尾相接的循环等待资源关系(互斥)

这是造成死锁必须要达到的4个条件，如果要避免死锁，只需要不满足其中某一个条件即可。而其中前3个条件是作为锁要符合的条件，所以要避免死锁就需要打破第4个条件，不出现循环等待锁的关系。

在开发过程中:

1.要注意加锁顺序，保证每个线程按同样的顺序进行加锁

2.要注意加锁时限，可以针对锁设置一个超时时间

3.要注意死锁检查，这是一种预防机制，确保在第一时间发现死锁并进行解决

## Synchronized与Lock锁的区别

|  |  |
| --- | --- |
| synchronized | Lock |
| 关键字 | 类 |
| 自动加锁和释放锁 | 需要手动调用unlock方法释放锁 |
| jvm层面的锁 | API层面的锁 |
| 非公平锁 | 可以选择公平或者非公平锁 |
| 锁是一个对象,并且锁的信息保存在了对象中 | 代码中通过int类型的state标识 |
| 有一个锁升级的过程 | 无 |

## 深拷贝与浅拷贝的理解

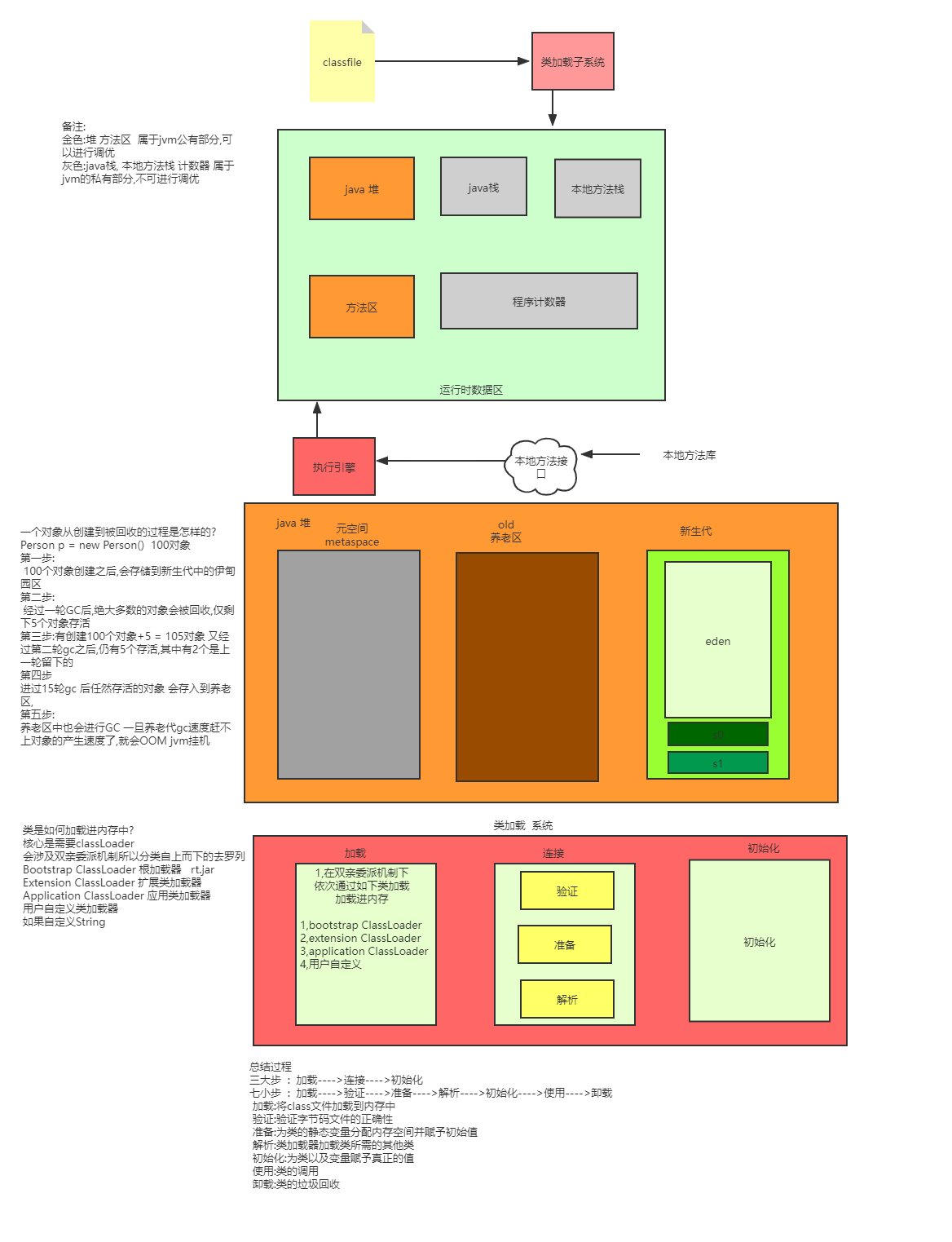
深拷贝和浅拷贝就是指对象的拷贝，一个对象中存在两种类型的属性，一种是基本数据类型，一种是实例对象的引用。

1.浅拷贝是指，只会拷贝基本数据类型的值，以及实例对象的引用地址，并不会复制一份引用地址所指向的对象，也就是浅拷贝出来的对象，内部的类属性指向的是同一个对象

2.深拷贝是指，既会拷贝基本数据类型的值，也会针对实例对象的引用地址所指向的对象进行复制，深拷贝出来的对象，内部的类执行指向的不是同一个对象

# Java高级篇

## JVM掌握一张图



## 为什么要使用线程池

线程池做的工作主要是控制运行的线程的数量，处理过程中将任务放入队列，然后在线程创建后启动这些任务，如果线程数量超过了最 大数量超出数量的线程排队等候，等其它线程执行完毕，再从队列中取出任务来执行。

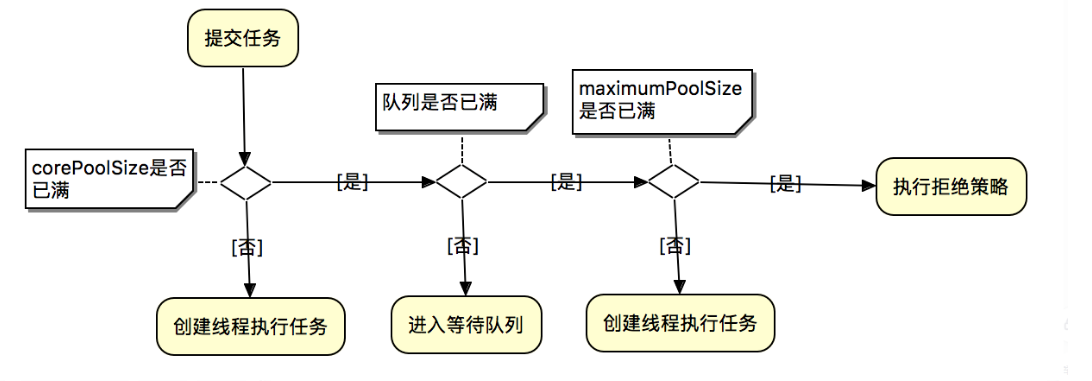
主要特点:线程复用;控制最大并发数:管理线程。

第一:降低资源消耗。通过重复利用己创建的线程降低线程创建和销毁造成的消耗。

第二:提高响应速度。当任务到达时，任务可以不需要的等到线程创建就能立即执行。

第三:提高线程的可管理性。线程是稀缺资源，如果无限制的创建，不仅会消耗系统资源，还会降低系统的稳定性，使用线程池可以进 行统一的分配，调优和监控

## 线程池底层工作原理



第一步：线程池刚创建的时候，里面没有任何线程，等到有任务过来的时候才会创建线程。当然也可以调用 prestartAllCoreThreads() 或者 prestartCoreThread() 方法预创建corePoolSize个线程

第二步：调用execute()提交一个任务时，如果当前的工作线程数<corePoolSize，直接创建新的线程执行这个任务

第三步：如果当时工作线程数量>=corePoolSize，会将任务放入任务队列中缓存

第四步：如果队列已满，并且线程池中工作线程的数量<maximumPoolSize，还是会创建线程执行这个任务

第五步：如果队列已满，并且线程池中的线程已达到maximumPoolSize，这个时候会执行拒绝策略，JAVA线程池默认的策略是AbortPolicy，即抛出RejectedExecutionException异常

## 如何自定义线程池



## 核心线程数如何确定

CPU密集型

CPU密集的意思是该任务需要大量的运算，而没有阻塞，CPU一直全速运行。CPU密集任务只有在真正的多核CPU上才可能得到加速(通过多线程)，而在单核CPU上，无论你开几个模拟的多线程该任务都不可能得到加速，因为CPU总的运算能力就那些。

CPU密集型任务配置尽可能少的线程数量:一般公式:CPU核数+1个线程的线程池

IO 密集型

由于IO密集型任务线程并不是一直在执行任务，则应配置尽可能多的线程，如CPU核数\*2     I0密集型，即该任务需要大量的I0，即大量的阻塞。在单线程上运行I0密集型的任务会导致浪费大量的CPU运算能力浪费在等待。所以在IO密集型任务中使用多线程可以大大的加速程序运行，即使在单核CPU上，这种加速主要就是利用了被浪费掉的阻塞时间。

I0密集型时，cpu核数 / (1-0.9)，核数为4的话，一般设置 40

## 拒绝策略了解多少?

|  |  |  |
| --- | --- | --- |
| 拒绝策略分类 | 含义 | 使用场景 |
| AbortPolicy | 丢弃任务并抛出异常 RejectedExecutionException | 我们项目中关于线程池的定义,使用的就是默认的  如果这种需求是关键的业务,eg:商品详情/购物车/首页 |
| DiscardPolicy | 安静的丢弃任务但是不抛出异常 | 设计的时候,一些无关紧要的业务可以采用此策略  Eg:单纯的展示某一项数据的情况 文章的浏览量/点赞个数 |
| DiscardOldestPolicy | 丢弃队列最前面的任务,然后重新提交被拒绝的任务 | 喜新厌旧  使用场景不多,可根据特定场景使用 |
| CallerRunsPolicy | 由调用线程处理该任务 | 使用场景非常少 |

# 第四章 MySQL数据库篇

## 1、数据库的存储引擎有哪些,有什么区别?

|  |  |
| --- | --- |
| 引擎名称 | 特点 |
| innoDB | 数据库默认的存储引擎 被设计用来处理大数据量的短期事务,除非有特别的要求,  否则我们使用默认即可 |
| myisam | 1,主要体现的功能storage(存储)  2,对事务安全性要求不高 读多写少 常被用于设计日志记录表  3,不支持事务,意味着存储效率高,但是文件一旦丢失,将无法恢复 |
| ARCHIVE | 档案存储引擎,只支持 insert 与select 操作 |
| BLACKHOLE  黑洞 | 1,没有实现任何的存储机制 它会丢失所有插入的数据,不做任何存储  2,但是服务器会记录blackhole的日志  3,可以用于复制数据到备份数据库或者简单的日志记录 |
| csv | 1,可以将普通的csv格式的文件作为mysql的表来处理  2,不支持所有,可以作为一种数据交换的机制而存在 |
| memory | 1,存储到内存而非磁盘  2,常被用于临时表 (用完即删)  3,如果需要快速访问数据, 并且这些数据不会被修改,重启以后丢失也没有关系,  可以使用该存储引擎 |

## 2、有没有设计过数据表?你是如何设计的?

|  |  |
| --- | --- |
| 第一范式 | 每一列属性(字段)不可分割的,字段必须保证原子性  两列的属性值相近或者一样的,尽量合并到一列或者分表,确保数据不冗余 |
| 第二范式 | 每一行的数据只能与其中一行有关 即 主键  一行数据只能做一件事情或者表达一个意思,  只要数据出现重复,就要进行表的拆分 |
| 第三范式 | 数据不能存在传递关系,每个属性都跟主键有直接关联而不是间接关联  IMG_256 |

## 聚簇索引与非聚簇索引有什么区别

都是B+树的数据结构

聚簇索引:将数据存储与索引放到了一块、并且是按照一定的顺序组织的，找到索引也就找到了数据，数据的物理存放顺序与索引顺序是一致的，即:只要索引是相邻的，那么对应的数据一定也是相邻地存放在磁盘上的

非聚簇索引叶子节点不存储数据、存储的是数据行地址，也就是说根据索引查找到数据行的位置再取磁盘查找数据，这个就有点类似一本书的目录，比如我们要找第三章第一节，那我们先在这个目录里面找，找到对应的页码后再去对应的页码看文章。

优势:

1、查询通过聚簇索引可以直接获取数据，相比非聚簇索引需要第二次查询(非覆盖索引的情况下)效率要高

2、聚簇索引对于范围查询的效率很高，因为其数据是按照大小排列的

3、聚簇索引适合用在排序的场合，非聚簇索引不适合

劣势;

1、维护索引很昂贵，特别是插入新行或者主键被更新导至要分页(pagesplit)的时候。建议在大量插入新行后，选在负载较低的时间段，通过OPTIMIZETABLE优化表，因为必须被移动的行数据可能造成碎片。使用独享表空间可以弱化碎片

2、表因为使用uuId(随机ID)作为主键，使数据存储稀疏，这就会出现聚簇索引有可能有比全表扫面更慢，所以建议使用int的auto\_increment作为主键

3、如果主键比较大的话，那辅助索引将会变的更大，因为辅助索引的叶子存储的是主键值，过长的主键值，会导致非叶子节点占用占用更多的物理空间

## B+tree 与 B-tree区别

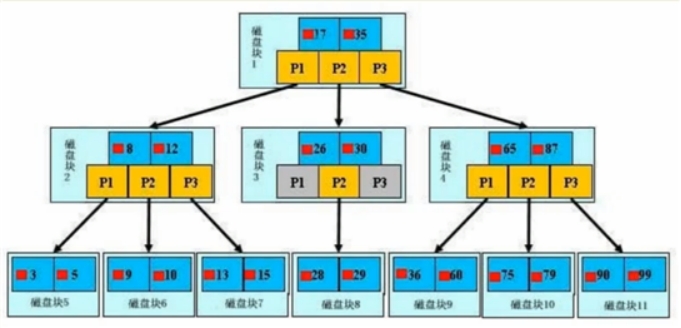
原理:分批次的将磁盘块加载进内存中进行检索,若查到数据,则直接返回,若查不到,则释放内存,并重新加载同等数据量的索引进内存,重新遍历

结构: 数据  向下的指针 指向数据的指针

特点:

        1，节点排序

        2 .一个节点了可以存多个元索，多个元索也排序了



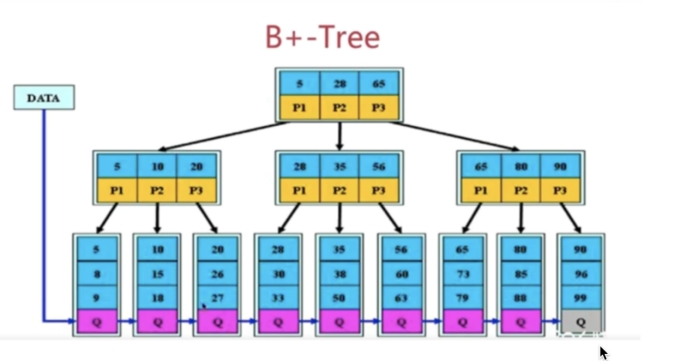
结构: 数据  向下的指针

特点:

    1.拥有B树的特点

    2.叶子节点之间有指针

    3.非叶子节点上的元素在叶子节点上都冗余了，也就是叶子节点中存储了所有的元素，并且排好顺序



从结构上看,B+Tree 相较于 B-Tree 而言  缺少了指向数据的指针 也就红色小方块;

Mysq|索引使用的是B+树，因为索引是用来加快查询的，而B+树通过对数据进行排序所以是可以提高查询速度的，然后通过一个节点中可以存储多个元素，从而可以使得B+树的高度不会太高，在Mysql中一个Innodb页就是一个B+树节点，一个Innodb页默认16kb，所以一般情况下一颗两层的B+树可以存2000万行左右的数据，然后通过利用B+树叶子节点存储了所有数据并且进行了排序，并且叶子节点之间有指针，可以很好的支持全表扫描，范围查找等SQL语句

## 以mysql为例 linux如何排查问题?

类似提问方式:如果线上环境出现问题比如网站卡顿重则瘫痪 如何是好?

--->linux--->mysql/redis/nacos/sentinel/sluth--->可以从以上提到的技术点中选择一个自己熟悉单技术点进行分析

以mysql为例

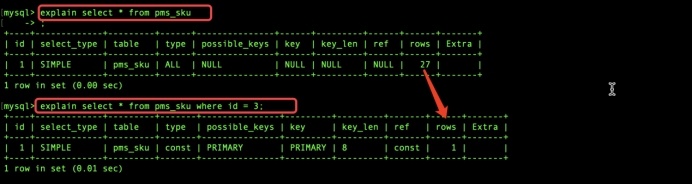
1,架构层面 是否使用主从

2,表结构层面 是否满足常规的表设计规范(大量冗余字段会导致查询会变得很复杂)

3,sql语句层面(⭐)

前提:由于慢查询日志记录默认是关闭的,所以开启数据库mysql的慢查询记录 的功能 从慢查询日志中去获取哪些sql语句时慢查询  默认10S ,从中获取到sql语句进行分析

3.1 explain 分析一条sql



Id:执行顺序 如果单表的话,无参考价值 如果是关联查询,会据此判断主表 从表

Select\_type:simple

Table:表

Type:  ALL 未创建索引 、const、 常量ref其他索引 、eq\_ref 主键索引、

Possible\_keys

Key  实际是到到索引到字段

Key\_len  索引字段数据结构所使用长度 与是否有默认值null 以及对应字段到数据类型有关，有一个理论值 有一个实际使用值也即key\_len的值

Rows  检索的行数 与查询返回的行数无关

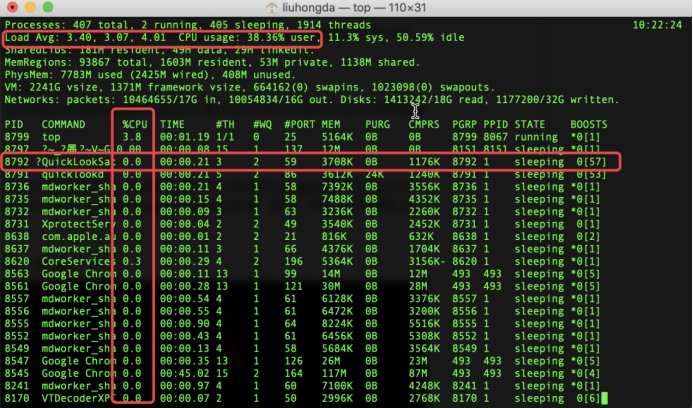
Extra  常见的值：usingfilesort 使用磁盘排序算法进行排序，事关排序 分组 的字段是否使用索引的核心参考值

还可能这样去提问：sql语句中哪些位置适合建索引/索引建立在哪个位置

Select id,name,age from user where id=1 and name=”xxx” order by age

总结:  查询字段  查询条件(最常用)   排序/分组字段

补充:如何判断是数据库的问题?可以借助于top命令



## 如何处理慢查询

在业务系统中，除了使用主键进行的查询，其他的都会在测试库上测试其耗时，慢查询的统计主要由运维在做，会定期将业务中的慢查询反馈给我们。

慢查询的优化首先要搞明白慢的原因是什么?是查询条件没有命中索引?是加载了不需要的数据列?还是数据量太大?

所以优化也是针对这三个方向来的

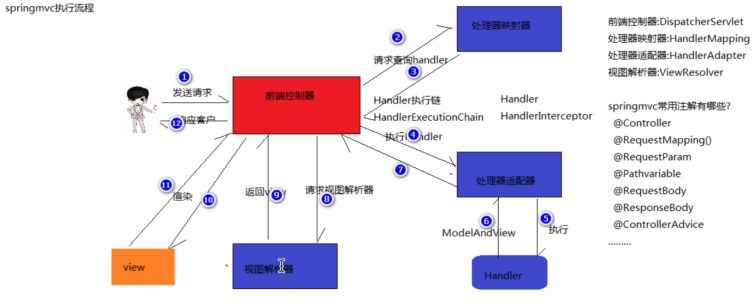
首先分析语句，看看是否加载了额外的数据，可能是查询了多余的行并且抛弃掉了，可能是加载了许多结果中并不需要的列，对语句进行分析以及重写。

分析语句的执行计划，然后获得其使用索引的情况，之后修改语句或者修改索引，使得语句可以尽可能的命中索引。

如果对语句的优化已经无法进行，可以考虑表中的数据量是否太大，如果是的话可以进行横向或者纵向的分表。

# 第五章 Java框架篇

## 1、简单的谈一下SpringMVC的工作流程？



1、用户向服务器发送请求，请求被SpringMVC的前端控制器DispatcherServlet截获。

2、DispatcherServlet对请求的URL（统一资源定位符）进行解析，得到URI(请求资源标识符)，然后根据该URI，调用HandlerMapping获得该Handler配置的所有相关的对象，包括Handler对象以及Handler对象对应的拦截器，这些对象都会被封装到一个HandlerExecutionChain对象当中返回。

3、DispatcherServlet根据获得的Handler，选择一个合适的HandlerAdapter。HandlerAdapter的设计符合面向对象中的单一职责原则，代码结构清晰，便于维护，最为重要的是，代码的可复制性高。HandlerAdapter会被用于处理多种Handler，调用Handler实际处理请求的方法。

4、提取请求中的模型数据，开始执行Handler(Controller)。在填充Handler的入参过程中，根据配置，spring将帮助做一些额外的工作

消息转换：将请求的消息，如json、xml等数据转换成一个对象，将对象转换为指定的响应信息。

数据转换：对请求消息进行数据转换，如String转换成Integer、Double等。

数据格式化：对请求的消息进行数据格式化，如将字符串转换为格式化数字或格式化日期等。

数据验证：验证数据的有效性如长度、格式等，验证结果存储到BindingResult或Error中。

5、Handler执行完成后，向DispatcherServlet返回一个ModelAndView对象，ModelAndView对象中应该包含视图名或视图模型。

6、根据返回的ModelAndView对象，选择一个合适的ViewResolver(视图解析器)返回给DispatcherServlet。

7、ViewResolver结合Model和View来渲染视图。

8、将视图渲染结果返回给客户端。

以上8个步骤，DispatcherServlet、HandlerMapping、HandlerAdapter和ViewResolver等对象协同工作，完成SpringMVC请求—>响应的整个工作流程，这些对象完成的工作对于开发者来说都是不可见的，开发者并不需要关心这些对象是如何工作的，开发者，只需要在Handler(Controller)当中完成对请求的业务处理。

## 2、Spring循环依赖

### 常见问法

请解释一下spring中的三级缓存

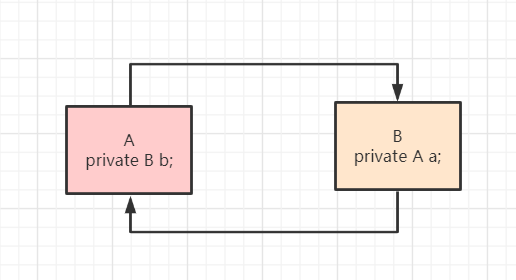
三级缓存分别是什么?三个Map有什么异同?

什么是循环依赖?请你谈谈?看过spring源码吗?

如何检测是否存在循环依赖?实际开发中见过循环依赖的异常吗?

多例的情况下,循环依赖问题为什么无法解决?

### 什么是循环依赖?



### 两种注入方式对循环依赖的影响?

官方解释

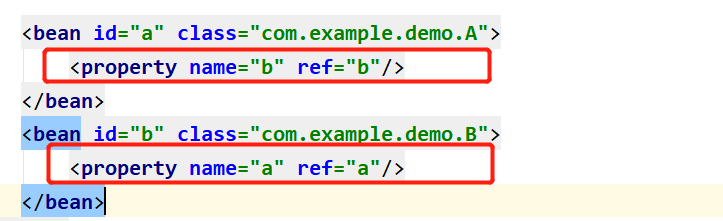
[https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-dependency-resolution](https://docs.spring.io/spring-framework/docs/current/reference/html/core.html" \l "beans-dependency-resolution)

### 相关概念

实例化:堆内存中申请空间

IMG_258

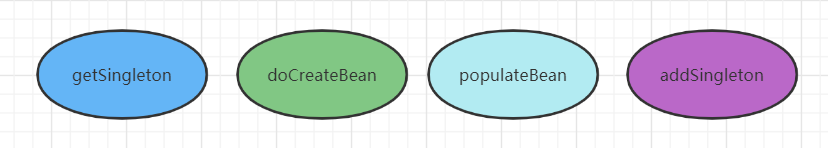
初始化:对象属性赋值

IMG_257

### 三级缓存

|  |  |  |
| --- | --- | --- |
| 名称 | 对象名 | 含义 |
| 一级缓存 | singletonObjects | 存放已经经历了完整生命周期的Bean对象 |
| 二级缓存 | earlySingletonObjects | 存放早期暴露出来的Bean对象，Bean的生命周期未结束（属性还未填充完) |
| 三级缓存 | singletonFactories | 存放可以生成Bean的工厂 |

### 四个关键方法



package org.springframework.beans.factory.support;

...

public class DefaultSingletonBeanRegistry extends SimpleAliasRegistry implements SingletonBeanRegistry {

...

/\*\*

单例对象的缓存:bean名称—bean实例，即:所谓的单例池。

表示已经经历了完整生命周期的Bean对象

第一级缓存

\*/

private final Map<String, Object> singletonObjects = new ConcurrentHashMap<>(256);

/\*\*

早期的单例对象的高速缓存: bean名称—bean实例。

表示 Bean的生命周期还没走完（Bean的属性还未填充）就把这个 Bean存入该缓存中也就是实例化但未初始化的 bean放入该缓存里

第二级缓存

\*/

private final Map<String, Object> earlySingletonObjects = new HashMap<>(16);

/\*\*

单例工厂的高速缓存:bean名称—ObjectFactory

表示存放生成 bean的工厂

第三级缓存

\*/

private final Map<String, ObjectFactory<?>> singletonFactories = new HashMap<>(16);

}

### debug源代码过程

需要22个断点(可选)

1，A创建过程中需要B，于是A将自己放到三级缓里面，去实例化B

2，B实例化的时候发现需要A，于是B先查一级缓存，没有，再查二级缓存，还是没有，再查三级缓存，找到了A然后把三级缓存里面的这个A放到二级缓存里面，并删除三级缓存里面的A

3，B顺利初始化完毕，将自己放到一级缓存里面(此时B里面的A依然是创建中状态)

然后回来接着创建A，此时B已经创建结束，直接从一级缓存里面拿到B，然后完成创建，并将A自己放到一级缓存里面。

### 总结

1，Spring创建 bean主要分为两个步骤，创建原始bean对象，接着去填充对象属性和初始化。

2，每次创建 bean之前，我们都会从缓存中查下有没有该bean，因为是单例，只能有一个。

3，当创建 A的原始对象后，并把它放到三级缓存中，接下来就该填充对象属性了，这时候发现依赖了B，接着就又去创建B，同样的流程，创建完B填充属性时又发现它依赖了A又是同样的流程，不同的是：这时候可以在三级缓存中查到刚放进去的原始对象A。

所以不需要继续创建，用它注入 B，完成 B的创建既然 B创建好了，所以 A就可以完成填充属性的步骤了，接着执行剩下的逻辑，闭环完成

Spring解决循环依赖依靠的是Bean的"中间态"这个概念，而这个中间态指的是已经实例化但还没初始化的状态—>半成品。实例化的过程又是通过构造器创建的，如果A还没创建好出来怎么可能提前曝光，所以构造器的循环依赖无法解决

### 其他衍生问题

问题1:为什么构造器注入属性无法解决循环依赖问题?

       由于spring中的bean的创建过程为先实例化 再初始化(在进行对象实例化的过程中不必赋值)将实例化好的对象暴露出去,供其他对象调用,然而使用构造器注入,必须要使用构造器完成对象的初始化的操作,就会陷入死循环的状态

问题2:一级缓存能不能解决循环依赖问题? 不能

       在三个级别的缓存中存储的对象是有区别的 一级缓存为完全实例化且初始化的对象 二级缓存实例化但未初始化对象 如果只有一级缓存,如果是并发操作下,就有可能取到实例化但未初始化的对象,就会出现问题

问题3:二级缓存能不能解决循环依赖问题?

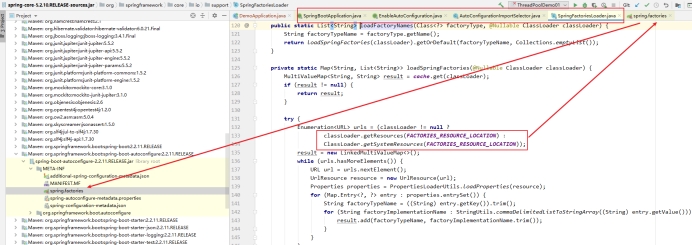
      理论上二级缓存可以解决循环依赖问题,但是需要注意,为什么需要在三级缓存中存储匿名内部类(ObjectFactory),原因在于 需要创建代理对象  eg:现有A类,需要生成代理对象 A是否需要进行实例化(需要) 在三级缓存中存放的是生成具体对象的一个匿名内部类,该类可能是代理类也可能是普通的对象,而使用三级缓存可以保证无论是否需要是代理对象,都可以保证使用的是同一个对象,而不会出现,一会儿使用普通bean 一会儿使用代理类

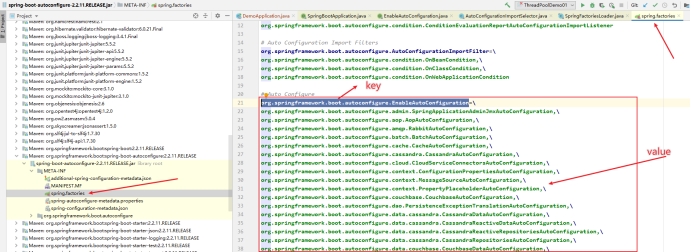
## 3、简述SpringBoot自动装配原理

第一个关键点：run方法底层原码有一个方法如下图



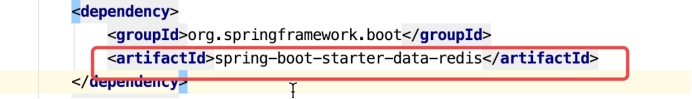
第二个关键点：启动类注解@SpringBootApplication





以redis为例

1,引入依赖规范 spring-boot-starter-data-XXX



2,springboot是如何通过场景启动器进行加载的?

 设计springboot加载核心技术点:自动装配 AutoConfiguration



总结:

Springboot在启动的时候会调用run方法，run方法会执行refreshContext()方法刷新容器，会在类路径下找到springboot-boot-autoconfigure/springboot-boot-autoconfigure.jar/META-INF/spring-factories文件，该文件中记录中众多的自动配置类，容器会根据我们是否引入依赖是否书写配置文件的情况，将满足条件的Bean注入到容器中，于是就实现了springboot的自动装配

## MyBatis中 #{}和${}的区别是什么？

#{}是预编译处理，${}是字符串替换；

Mybatis在处理#{}时，会将sql中的#{}替换为?号，调用PreparedStatement的set方法来赋值；

Mybatis在处理${}时，就是把${}替换成变量的值；

使用#{}可以有效的防止SQL注入，提高系统安全性。

## 谈谈你对Spring 的理解

Spring 是一个开源框架，为简化企业级应用开发而生。Spring 可以是使简单的JavaBean 实现以前只有EJB 才能实现的功能。Spring 是一个 IOC 和 AOP 容器框架。

Spring 容器的主要核心是：

控制反转（IOC），传统的 java 开发模式中，当需要一个对象时，我们会自己使用 new 或者 getInstance 等直接或者间接调用构造方法创建一个对象。而在 spring 开发模式中，spring 容器使用了工厂模式为我们创建了所需要的对象，不需要我们自己创建了，直接调用spring 提供的对象就可以了，这是控制反转的思想。

依赖注入（DI），spring 使用 javaBean 对象的 set 方法或者带参数的构造方法为我们在创建所需对象时将其属性自动设置所需要的值的过程，就是依赖注入的思想。

面向切面编程（AOP），在面向对象编程（oop）思想中，我们将事物纵向抽成一个个的对象。而在面向切面编程中，我们将一个个的对象某些类似的方面横向抽成一个切面，对这个切面进行一些如权限控制、事物管理，记录日志等公用操作处理的过程就是面向切面编程的思想。AOP 底层是动态代理，如果是接口采用 JDK 动态代理，如果是类采用CGLIB 方式实现动态代理。

## 5、Spring中常用的设计模式

（1）代理模式——spring 中两种代理方式，若目标对象实现了若干接口，spring 使用jdk 的java.lang.reflect.Proxy类代理。若目标兑现没有实现任何接口，spring 使用 CGLIB 库生成目标类的子类。

（2）单例模式——在 spring 的配置文件中设置 bean 默认为单例模式。

（3）模板方式模式——用来解决代码重复的问题。

比如：RestTemplate、JmsTemplate、JpaTemplate

（4）工厂模式——在工厂模式中，我们在创建对象时不会对客户端暴露创建逻辑，并且是通过使用同一个接口来指向新创建的对象。Spring 中使用 beanFactory 来创建对象的实例。

## 6、介绍一下Spring bean 的生命周期、注入方式和作用域

**Bean的生命周期**

（1）默认情况下，IOC容器中bean的生命周期分为五个阶段:

* 调用构造器 或者是通过工厂的方式创建Bean对象
* 给bean对象的属性注入值
* 调用初始化方法，进行初始化， 初始化方法是通过init-method来指定的.
* 使用
* IOC容器关闭时， 销毁Bean对象.

（2）当加入了Bean的后置处理器后，IOC容器中bean的生命周期分为七个阶段:

* 调用构造器 或者是通过工厂的方式创建Bean对象
* 给bean对象的属性注入值
* 执行Bean后置处理器中的 postProcessBeforeInitialization
* 调用初始化方法，进行初始化， 初始化方法是通过init-method来指定的.
* 执行Bean的后置处理器中 postProcessAfterInitialization
* 使用
* IOC容器关闭时， 销毁Bean对象

只需要回答出第一点即可，第二点也回答可适当 加分。

**注入方式：**

通过 setter 方法注入

通过构造方法注入

**Bean的作用域**

总共有四种作用域:

* Singleton 单例的
* Prototype 原型的
* Request
* Session

# 第六章Redis数据库篇

## 1、简单介绍一下redis/为什么不直接操作内存呢?

【答案解析】

   （1）redis是一个key-value类型的非关系型数据库，基于内存也可持久化的数据库，相对于关系型数据库（数据主要存在硬盘中），性能高，因此我们一般用redis来做缓存使用；并且redis支持丰富的数据类型，比较容易解决各种问题

    （2）Redis的Value支持5种数据类型，string、hash、list、set、zset（sorted set）;

以下为5中类型比较经典的使用场景

|  |  |
| --- | --- |
| 类型 | 使用场景 |
| string | String类型是最简单的类型，一个key对应一个value，项目中我们主要利用单点登录中的token用string类型来存储；商品详情 |
| hash | Hash类型中的key是string类型，value又是一个map（key-value），针对这种数据特性，比较适合存储对象，在我们项目中由于购物车是用redis来存储的，因此选择redis的散列（hash）来存储； |
| list | List类型是按照插入顺序的字符串链表（双向链表），主要命令是LPOP和RPUSH，能够支持反向查找和遍历，如果使用的话主要存储商品评论列表，key是该商品的ID，value是商品评论信息列表；消息队列 |
| set | Set类型是用哈希表类型的字符串序列，没有顺序，集合成员是唯一的，没有重复数据，底层主要是由一个value永远为null的hashmap来实现的。  可以基于 Set 玩儿交集、并集、差集的操作，比如交集吧，我们可以把两个人的好友列表整一个交集，看看俩人的共同好友是谁？ |
| zset | zset（sorted set）类型和set类型基本是一致的，不同的是zset这种类型会给每个元素关联一个**double**类型的分数（score），这样就可以为成员排序，并且插入是有序的。这种数据类型如果使用的话主要用来统计商品的销售排行榜，比如：items:sellsort 10 1001 20 1002 这个代表编号是1001的商品销售数量为10,编号为1002的商品销售数量为20/附件的人 |

## 2、单线程的redis为什么读写速度快?

1. 纯内存操作
2. 单线程操作，避免了频繁的上下文切换
3. 采用了非阻塞I/O多路复用机制

## 3、redis为什么是单线程的?

官方FAQ表示，因为Redis是基于内存的操作，CPU不是Redis的瓶颈，Redis的瓶颈最有可能是机器内存的大小或者网络带宽。既然单线程容易实现，而且CPU不会成为瓶颈，那就顺理成章地采用单线程的方案了Redis利用队列技术将并发访问变为串行访问

1）绝大部分请求是纯粹的内存操作

2）采用单线程,避免了不必要的上下文切换和竞争条件

## 4、redis服务器的的内存是多大?

配置文件中设置redis内存的参数：。

该参数如果不设置或者设置为0，则redis默认的内存大小为：

32位下默认是3G

64位下不受限制

一般推荐Redis设置内存为最大物理内存的四分之三，也就是0.75

命令行设置config set maxmemory <内存大小，单位字节>，服务器重启失效

config get maxmemory获取当前内存大小

永久则需要设置maxmemory参数，maxmemory是bytes字节类型，注意转换

## 5、为什么Redis的操作是原子性的，怎么保证原子性的？

对于Redis而言，命令的原子性指的是：一个操作的不可以再分，操作要么执行，要么不执行。

Redis的操作之所以是原子性的，是因为Redis是单线程的。

Redis本身提供的所有API都是原子操作，Redis中的事务其实是要保证批量操作的原子性。

多个命令在并发中也是原子性的吗？

不一定， 将get和set改成单命令操作，incr 。使用Redis的事务，或者使用Redis+Lua==的方式实现.

## 6、你还用过其他的缓存吗？这些缓存有什么区别？都在什么场景下去用？

对于缓存了解过redis和memcache，redis我们在项目中用的比较多，memcache没用过，但是了解过一点；

Memcache和redis的区别：

|  |  |  |
| --- | --- | --- |
| 比较项 | reids | memcache |
| 存储方式 | redis可以持久化其数据 | Memecache把数据全部存在内存之中，断电后会挂掉，数据不能超过内存大小 |
| 数据支持 | 支持丰富的数据类型,提供list，set，zset，hash等数据结构的存储 | memcached所有的值均是简单的字符串 |
| 底层模型 | Redis直接自己构建了VM 机制 | 无 |
| value值大小 | Redis 最大可以达到 512M | value 不能超过 1M 字节 |
| 速度 | Redis 采用单线程模式处理请求。这样做的原因有 2 个：一个是因为采用了非阻塞的异步事件处理机制；另一个是缓存数据都是内存操作 IO 时间不会太长，单线程可以避免线程上下文切换产生的代价 | MC 处理请求时使用多线程异步 IO 的方式，可以合理利用 CPU 多核的优势，性能非常优秀 |
| 数据备份 | Redis支持数据的备份，即master-slave模式的数据备份,能够提供高可用服务 | 当容量存满时，会对缓存中的数据进行剔除，剔除时除了会对过期 key 进行清理，还会按 LRU 策略对数据进行剔除。 |
| 应用场景 | 适用于对读写效率要求高、数据处理业务复杂、安全性要求较高的系统 | 适合多读少写，大数据量的情况（一些官网的文章信息等） |

## 7、Redis在你们项目中是怎么用的？

【答案解析】

（1）门户(首页)系统中的首页内容信息的展示。（商品类目、广告、热门商品等信息）门户系统的首页是用户访问量最大的，而且这些数据一般不会经常修改，因此为了提高用户的体验，我们选择将这些内容放在缓存中；

（2）单点登录系统中也用到了redis。因为我们是分布式系统，存在session之间的共享问题，因此在做单点登录的时候，我们利用redis来模拟了session的共享，来存储用户的信息，实现不同系统的session共享；

（3）我们项目中同时也将购物车的信息设计存储在redis中，购物车在数据库中没有对应的表，用户登录之后将商品添加到购物车后存储到redis中，key是用户id，value是购物车对象；

（4）因为针对评论这块，我们需要一个商品对应多个用户评论，并且按照时间顺序显示评论，为了提高查询效率，因此我们选择了redis的list类型将商品评论放在缓存中；

（5）在统计模块中，我们有个功能是做商品销售的排行榜，因此选择redis的zset结构来实现；

还有一些其他的应用场景，主要就是用来作为缓存使用。

## 8、对redis的持久化了解不？

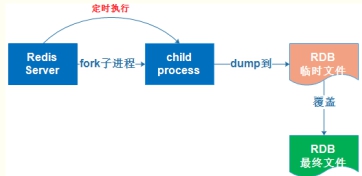
Redis是内存型数据库，同时它也可以持久化到硬盘中，redis的持久化方式有两种：

（1）RDB（**半持久化方式**）：

按照配置不定期的通过异步的方式、快照的形式直接把内存中的数据持久化到磁盘的一个dump.rdb文件（二进制文件）中；

这种方式是redis默认的持久化方式，它在配置文件（redis.conf）中的格式是：save N M，表示的是在N秒之内发生M次修改，则redis抓快照到磁盘中；

原理：当redis需要持久化的时候，redis会fork一个子进程，这个子进程会将数据写到一个临时文件中；当子进程完成写临时文件后，会将原来的.rdb文件替换掉，这样的好处是写时拷贝技术（copy-on-write），可以参考下面的流程图；



优点：只包含一个文件，对于文件备份、灾难恢复而言，比较实用。因为我们可以轻松的将一个单独的文件转移到其他存储媒介上；性能最大化，因为对于这种半持久化方式，使用的是写时拷贝技术，可以极大的避免服务进程执行IO操作；相对于AOF来说，如果数据集很大，RDB的启动效率就会很高

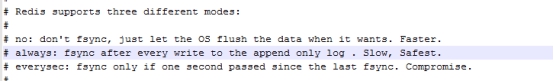
缺点：如果想保证数据的高可用（最大限度的包装数据丢失），那么RDB这种半持久化方式不是一个很好的选择，因为系统一旦在持久化策略之前出现宕机现象，此前没有来得及持久化的数据将会产生丢失；rdb是通过fork进程来协助完成持久化的，因此当数据集较大的时候，我们就需要等待服务器停止几百毫秒甚至一秒；

（2）AOF（全持久化的方式）

 把每一次数据变化都通过write()函数将你所执行的命令追加到一个appendonly.aof文件里面；

    Redis默认是不支持这种全持久化方式的，需要将no改成yes

实现文件刷新的三种方式：



no:不会自动同步到磁盘上，需要依靠OS（操作系统）进行刷新，效率快，但是安全性就比较差；

always:每提交一个命令都调用fsync刷新到aof文件，非常慢，但是安全；

everysec:每秒钟都调用fsync刷新到aof文件中，很快，但是可能丢失一秒内的数据，推荐使用，兼顾了速度和安全；

原理：redis需要持久化的时候，fork出一个子进程，子进程根据内存中的数据库快照，往临时文件中写入重建数据库状态的命令；父进程会继续处理客户端的请求，除了把写命令写到原来的aof中，同时把收到的写命令缓存起来，这样包装如果子进程重写失败的话不会出问题；当子进程把快照内容以命令方式写入临时文件中后，子进程会发送信号给父进程，父进程会把缓存的写命令写入到临时文件中；接下来父进程可以使用临时的aof文件替换原来的aof文件，并重命名，后面收到的写命令也开始往新的aof文件中追加。下面的图为最简单的方式，其实也是利用写时复制原则。

IMG_259

优点：

 数据安全性高

 该机制对日志文件的写入操作采用的是append模式，因此在写入过程中即使出现宕机问题，也不会破坏日志文件中已经存在的内容；

缺点：

对于数量相同的数据集来说，aof文件通常要比rdb文件大，因此rdb在恢复大数据集时的速度大于AOF；

根据同步策略的不同，AOF在运行效率上往往慢于RDB，每秒同步策略的效率是比较高的，同步禁用策略的效率和RDB一样高效；

针对以上两种不同的持久化方式，如果缓存数据安全性要求比较高的话，用aof这种持久化方式（比如项目中的购物车）；如果对于大数据集要求效率高的话，就可以使用默认的。而且这两种持久化方式可以同时使用。

## 9、redis的过期策略以及内存淘汰机制有了解过吗

redis采用的是定期删除+惰性删除策略。

为什么不用定时删除策略?

定时删除,用一个定时器来负责监视key,过期则自动删除。虽然内存及时释放，但是十分消耗CPU资源。在大并发请求下，CPU要将时间应用在处理请求，而不是删除key,因此没有采用这一策略.

定期删除+惰性删除是如何工作的呢?

定期删除，redis默认每个100ms检查，是否有过期的key,有过期key则删除。需要说明的是，redis不是每个100ms将所有的key检查一次，而是随机抽取进行检查(如果每隔100ms,全部key进行检查，redis岂不是卡死)。因此，如果只采用定期删除策略，会导致很多key到时间没有删除。

于是，惰性删除派上用场。也就是说在你获取某个key的时候，redis会检查一下，这个key如果设置了过期时间那么是否过期了？如果过期了此时就会删除。

采用定期删除+惰性删除就没其他问题了么?

不是的，如果定期删除没删除key。然后你也没即时去请求key，也就是说惰性删除也没生效。这样，redis的内存会越来越高。那么就应该采用内存淘汰机制。

在redis.conf中有一行配置

maxmemory-policy volatile-lru

该配置就是配内存淘汰策略的(什么，你没配过？好好反省一下自己)

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，新写入操作会报错

ps：如果没有设置 expire 的key, 不满足先决条件(prerequisites); 那么 volatile-lru, volatile-random 和 volatile-ttl 策略的行为, 和 noeviction(不删除) 基本上一致。

## 10、做过redis的集群吗？你们做集群的时候搭建了几台，都是怎么搭建的？

针对这类问题，我们首先考虑的是为什么要搭建集群？（这个需要针对我们的项目来说）

Redis的数据是存放在内存中的，这就意味着redis不适合存储大数据，大数据存储一般公司常用hadoop中的Hbase或者MogoDB。因此redis主要用来处理高并发的，用我们的项目来说，电商项目如果并发大的话，一台单独的redis是不能足够支持我们的并发，这就需要我们扩展多台设备协同合作，即用到集群。

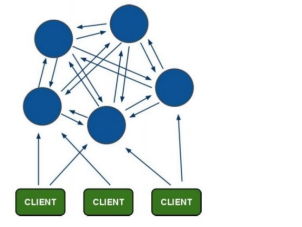
Redis搭建集群的方式有多种，例如：客户端分片、Twemproxy、Codis等，但是redis3.0之后就支持redis-cluster集群，这种方式采用的是无中心结构，每个节点保存数据和整个集群的状态，每个节点都和其他所有节点连接。如果使用的话就用redis-cluster集群。

集群这块直接说是公司运维搭建的，小公司的话也有可能由我们自己搭建，开发环境我们也可以直接用单机版的。但是可以了解一下redis的集群版。搭建redis集群的时候，对于用到多少台服务器，每家公司都不一样，大家针对自己项目的大小去衡量。举个简单的例子：

我们项目中redis集群主要搭建了6台，3主（为了保证redis的投票机制）3从（

【扩展】高可用），每个主服务器都有一个从服务器，作为备份机。

1、架构图如下：



（1）所有的节点都通过PING-PONG机制彼此互相连接；

（2）每个节点的fail是通过集群中超过半数的节点检测失效时才生效；

（3）客户端与redis集群连接，只需要连接集群中的任何一个节点即可；

（4）Redis-cluster把所有的物理节点映射到【0-16383】slot上，负责维护

2、容错机制（投票机制）

（1）选举过程是集群中的所有master都参与，如果半数以上master节点与故障节点连接超过时间，则认为该节点故障，自动会触发故障转移操作；

（2）集群不可用？

     a:如果集群任意master挂掉，并且当前的master没有slave，集群就会fail;

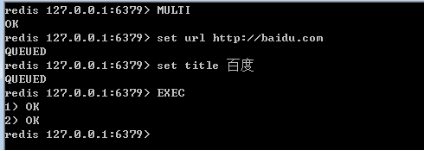
     b:如果集群超过半数以上master挂掉，无论是否有slave，整个集群都会fail;

## 11、说说Redis哈希槽的概念？

Redis集群没有使用一致性hash,而是引入了哈希槽的概念，Redis集群有16384个哈希槽，每个key通 过CRC16校验后对16384取模来决定放置哪个槽，集群的每个节点负责一部分hash槽。

## 12、redis有事务吗？

Redis是有事务的，redis中的事务是一组命令的集合，这组命令要么都执行，要不都不执行，redis事务的实现，需要用到MULTI（事务的开始）和EXEC（事务的结束）命令 ;



当输入MULTI命令后，服务器返回OK表示事务开始成功，然后依次输入需要在本次事务中执行的所有命令，每次输入一个命令服务器并不会马上执行，而是返回”QUEUED”，这表示命令已经被服务器接受并且暂时保存起来，最后输入EXEC命令后，本次事务中的所有命令才会被依次执行，可以看到最后服务器一次性返回了两个OK，这里返回的结果与发送的命令是按顺序一一对应的，这说明这次事务中的命令全都执行成功了。

Redis的事务除了保证所有命令要不全部执行，要不全部不执行外，还能保证一个事务中的命令依次执行而不被其他命令插入。同时，redis的事务是不支持回滚操作的。

【扩展】

  Redis的事务中存在一个问题，如果一个事务中的B命令依赖上一个命令A怎么办？

这会涉及到redis中的WATCH命令：可以监控一个或多个键，一旦其中有一个键被修改（或删除），之后的事务就不会执行，监控一直持续到EXEC命令（事务中的命令是在EXEC之后才执行的，EXEC命令执行完之后被监控的键会自动被UNWATCH）。

应用场景：待定

## 13、是否了解过redis的安全机制？

1、redis的安全机制（你们公司redis的安全这方面怎么考虑的？）

漏洞介绍：redis默认情况下，会绑定在bind 0.0.0.0:6379，这样就会将redis的服务暴露到公网上，如果在没有开启认证的情况下，可以导致任意用户在访问目标服务器的情况下未授权访问redis以及读取redis的数据，攻击者就可以在未授权访问redis的情况下可以利用redis的相关方法，成功在redis服务器上写入公钥，进而可以直接使用私钥进行直接登录目标主机；

比如：可以使用FLUSHALL方法，整个redis数据库将被清空

解决方案：

（1）禁止一些高危命令。修改redis.conf文件，用来禁止远程修改DB文件地址，比如 rename-command FLUSHALL "" 、rename-command CONFIG"" 、rename-command EVAL “”等；

（2）以低权限运行redis服务。为redis服务创建单独的用户和根目录，并且配置禁止登录；

（3）为redis添加密码验证。修改redis.conf文件，添加

requirepass mypassword；

（4）禁止外网访问redis。修改redis.conf文件，添加或修改 bind 127.0.0.1，使得redis服务只在当前主机使用；

（5）做log监控，及时发现攻击；

（6）服务器不安装

## 14、你对redis的哨兵机制了解多少？（redis2.6以后出现的）

哨兵机制：

监控：监控主数据库和从数据库是否正常运行；

提醒：当被监控的某个redis出现问题的时候，哨兵可以通过API向管理员或者其他应用程序发送通知；

自动故障迁移：主数据库出现故障时，可以自动将从数据库转化为主数据库，实现自动切换；

具体的配置步骤面试中可以说参考的网上的文档。要注意的是，如果master主服务器设置了密码，记得在哨兵的配置文件（sentinel.conf）里面配置访问密码

## 15、redis缓存与mysql数据库之间的数据一致性问题?

不管先保存到MySQL，还是先保存到Redis都面临着一个保存成功而另外一个保存失败的情况。

不管是先写MySQL数据库，再删除Redis缓存；还是先删除缓存，再写库，都有可能出现数据不一致的情况。举一个例子：

1.如果删除了缓存Redis，还没有来得及写库MySQL，另一个线程就来读取，发现缓存为空，则去数据库中读取数据写入缓存，此时缓存中为脏数据。

2.如果先写了库，在删除缓存前，写库的线程宕机了，没有删除掉缓存，则也会出现数据不一致情况。

因为写和读是并发的，没法保证顺序,就会出现缓存和数据库的数据不一致的问题。

解决：

基于mysql的binlog日志（canal）  
消息队列(双删)

## 16、什么是redis的缓存穿透？如何防止穿透？

缓存穿透是指查询一个不存在的数据，由于缓存无法命中，将去查询数据库，但是数据库也无此记录，并且出于容错考虑，我们没有将这次查询的null写入缓存，这将导致这个不存在的数据每次请求都要到存储层去查询，失去了缓存的意义。在流量大时，可能DB就挂掉了，要是有人利用不存在的key频繁攻击我们的应用，这就是漏洞。

解决：空结果也进行缓存，但它的过期时间会很短，最长不超过五分钟。

## 17、什么是redis的缓存雪崩？如何防止？

缓存雪崩是指在我们设置缓存时采用了相同的过期时间，导致缓存在某一时刻同时失效，请求全部转发到DB，DB瞬时压力过重雪崩。

解决：原有的失效时间基础上增加一个随机值，比如1-5分钟随机，这样每一个缓存的过期时间的重复率就会降低，就很难引发集体失效的事件。

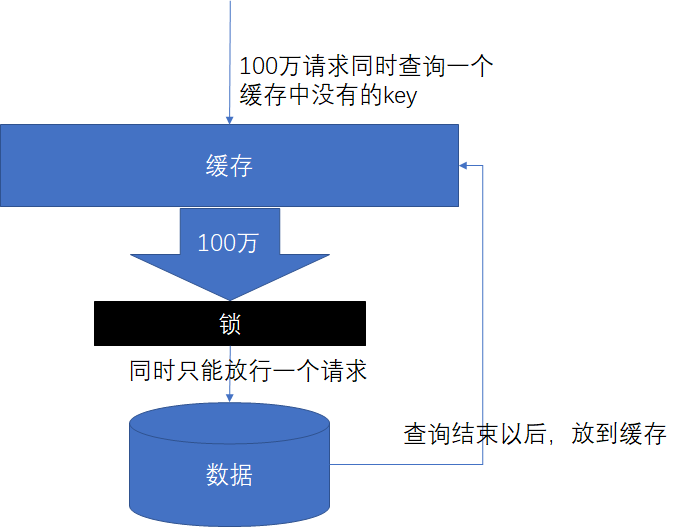
## 18、什么是redis的缓存击穿？如何防止？

缓存击穿是指对于一些设置了过期时间的key，如果这些key可能会在某些时间点被超高并发地访问，是一种非常“热点”的数据。这个时候，需要考虑一个问题：如果这个key在大量请求同时进来之前正好失效，那么所有对这个key的数据查询都落到db，我们称为缓存击穿。

与缓存雪崩的区别：

1. 击穿是一个热点key失效
2. 雪崩是很多key集体失效

解决：锁



## 19、redis中对于生存时间的应用

 Redis中可以使用expire命令设置一个键的生存时间，到时间后redis会自动删除；

  应用场景：

（1）设置限制的优惠活动的信息；

（2）一些及时需要更新的数据，积分排行榜；

（3）手机验证码的时间；

（4）限制网站访客访问频率；

# 第七章MQ消息队列

## 如何保证消息的可靠性传输/如何处理消息丢失问题?

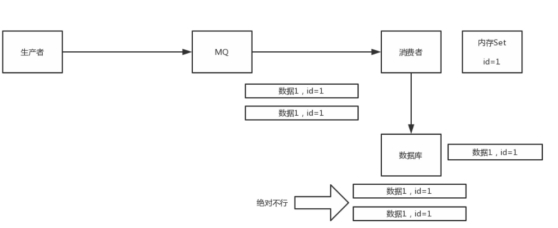
|  |  |
| --- | --- |
| 考虑维度 | 分析 |
| 生产者 | 原因：网络中断  解决1：可以使用rabbitmq提供的事务功能        就是生产者发送数据之前开启rabbitmq事务（channel.txSelect），然后发送消息，如果消息没有成功被rabbitmq接收到，那么生产者会收到异常报错，此时就可以回滚事务（channel.txRollback），然后重试发送消息；如果收到了消息，那么可以提交事务（channel.txCommit）此方法会严重降低系统的吞吐量，性能消耗太大  解决2：生产者开发confirm模式之后        你每次写的消息都会分配一个唯一的id，然后如果写入了rabbitmq中，rabbitmq会给你回传一个ack消息，告诉你说这个消息ok了。如果rabbitmq没能处理这个消息，会回调你一个nack接口，告诉你这个消息接收失败，你可以重试。而且你可以结合这个机制自己在内存里维护每个消息id的状态，如果超过一定时间还没接收到这个消息的回调，那么你可以重发。 |
| rabbitMQ本身 | 原因：MQ宕机  解决：开启rabbitmq的持久化       就是消息写入之后会持久化到磁盘，哪怕是rabbitmq自己挂了，            恢复之后会自动读取之前存储的数据，一般数据不会丢。除非极其罕见的是，rabbitmq还没持久化，自己就挂了，可能导致少量数据会丢失的，但是这个概率较小。  设置持久化有两个步骤，第一个是创建queue的时候将其设置为持久化的，这样就可以保证rabbitmq持久化queue的元数据，但是不会持久化queue里的数据；第二个是发送消息的时候将消息的deliveryMode设置为2，就是将消息设置为持久化的，此时rabbitmq就会将消息持久化到磁盘上去。必须要同时设置这两个持久化才行，rabbitmq哪怕是挂了，再次重启，也会从磁盘上重启恢复queue，恢复这个queue里的数据。  而且持久化可以跟生产者那边的confirm机制配合起来，只有消息被持久化到磁盘之后，才会通知生产者ack了，所以哪怕是在持久化到磁盘之前，rabbitmq挂了，数据丢了，生产者收不到ack，你也是可以自己重发的。 |
| 消费者 | 原因：刚消费到，还没处理，结果进程就挂了  解决：     用rabbitmq提供的ack机制，简单来说，就是你关闭rabbitmq自动ack，可以通过一个api来调用就行，然后每次你自己代码里确保处理完的时候，再程序里ack一把。这样的话，如果你还没处理完，不就没有ack？那rabbitmq就认为你还没处理完，这个时候rabbitmq会把这个消费分配给别的consumer去处理，消息是不会丢的。 |

## 如何保证消息不被重复消费?

面试官心理分析

**其实这个很常见的一个问题，这俩问题基本可以连起来问。既然是消费消息，那肯定要考虑考虑会不会重复消费？能不能避免重复消费？或者重复消费了也别造成系统异常可以吗？这个是MQ领域的基本问题，其实本质上还是问你使用消息队列如何保证幂等性，这个是你架构里要考虑的一个问题。**

一般情况下，任何一种消息队列中间件都会出现消息的重复消费问题，因为这个问题不是mq能够保证的，需要程序员结合实际业务场景来控制的，所以回答这个问题最好是结合实际业务场景阐述



（1）数据要写库操作，最好先根据主键查一下，如果这数据都有了，就不再执行insert操作，可以update

（2）写入redis，redis的set操作天然幂等性

（3）你需要让生产者发送每条数据的时候，里面加一个全局唯一的id，类似订单id之类的东西，然后你这里消费到了之后，先根据这个id去比如redis里查一下，之前消费过吗？如果没有消费过，你就处理，然后这个id写redis。如果消费过了，那你就别处理了，保证别重复处理相同的消息即可。（防止订单重复提交）

## 3、如何解决消息队列的延时以及过期失效问题?消息队列满了之后该如何处理?有几百万的消息持续积压几小时,说说如何解决?

##### 方案分析

该问题,其本质针对的场景，都是说，可能你的消费端出了问题，不消费了，或者消费的极其极其慢。另外还有可能你的消息队列集群的磁盘都快写满了，都没人消费，这个时候怎么办？或者是整个这就积压了几个小时，你这个时候怎么办？或者是你积压的时间太长了，导致比如rabbitmq设置了消息过期时间后就没了怎么办？

所以这种问题线上常见的，一般不出，一出就是大问题，一般常见于，举个例子，消费端每次消费之后要写mysql，结果mysql挂了，消费端挂掉了。导致消费速度极其慢。

分析1+话术

这个是我们真实遇到过的一个场景，确实是线上故障了，这个时候要不然就是修复consumer的问题，让他恢复消费速度，然后傻傻的等待几个小时消费完毕。(可行,但是不建议 在面试的时候说)

一个消费者一秒是1000条，一秒3个消费者是3000条，一分钟是18万条，1000多万条

所以如果你积压了几百万到上千万的数据，即使消费者恢复了，也需要大概1小时的时间才能恢复过来

一般这个时候，只能操作临时紧急扩容了，具体操作步骤和思路如下：

1）先修复consumer的问题，确保其恢复消费速度，然后将现有cnosumer都停掉

**2）新建一个topic，partition是原来的10倍，临时建立好原先10倍或者20倍的queue数量**

3）然后写一个临时的分发数据的consumer程序，这个程序部署上去消费积压的数据，消费之后不做耗时的处理，直接均匀轮询写入临时建立好的10倍数量的queue

4）接着临时征用10倍的机器来部署consumer，每一批consumer消费一个临时queue的数据

5）这种做法相当于是临时将queue资源和consumer资源扩大10倍，以正常的10倍速度来消费数据

6）等快速消费完积压数据之后，得恢复原先部署架构，重新用原先的consumer机器来消费消息

分析2+话术

rabbitmq是可以设置过期时间的，就是TTL，如果消息在queue中积压超过一定的时间就会被rabbitmq给清理掉，这个数据就没了。那这就是第二个坑了。这就不是说数据会大量积压在mq里，而是大量的数据会直接搞丢。

这个情况下，就不是说要增加consumer消费积压的消息，因为实际上没啥积压，而是丢了大量的消息。我们可以采取一个方案，就是批量重导，这个我们之前线上也有类似的场景干过。就是大量积压的时候，我们当时就直接丢弃数据了，然后等过了高峰期以后，比如大家一起喝咖啡熬夜到晚上12点以后，用户都睡觉了。

这个时候我们就开始写程序，将丢失的那批数据，写个临时程序，一点一点的查出来，然后重新灌入mq里面去，把白天丢的数据给他补回来。也只能是这样了。

假设1万个订单积压在mq里面，没有处理，其中1000个订单都丢了，你只能手动写程序把那1000个订单给查出来，手动发到mq里去再补一次

分析3+话术

如果走的方式是消息积压在mq里，那么如果你很长时间都没处理掉，此时导致mq都快写满了，咋办？这个还有别的办法吗？没有，谁让你第一个方案执行的太慢了，你临时写程序，接入数据来消费，消费一个丢弃一个，都不要了，快速消费掉所有的消息。然后走第二个方案，到了晚上再补数据吧。

## 4、如果让你写一个消息队列,该如何进行架构设计?说一下思路

面试官心理分析

（1）你有没有对某一个消息队列做过较为深入的原理的了解，或者从整体了解把握住一个mq的架构原理

（2）看看你的设计能力，给你一个常见的系统，就是消息队列系统，看看你能不能从全局把握一下整体架构设计，给出一些关键点出来

类似问题

如果让你来设计一个spring框架你会怎么做？如果让你来设计一个dubbo框架你会怎么做？如果让你来设计一个mybatis框架你会怎么做？

回答思路

（1）首先这个mq得支持可伸缩性吧，就是需要的时候快速扩容，就可以增加吞吐量和容量，那怎么搞？设计个分布式的系统

（2）其次你得考虑一下这个mq的数据要不要落地磁盘吧？那肯定要了，落磁盘，才能保证别进程挂了数据就丢了。那落磁盘的时候怎么落啊？顺序写，这样就没有磁盘随机读写的寻址开销，磁盘顺序读写的性能是很高的。

（3）其次你考虑一下你的mq的可用性啊？

（4）能不能支持数据0丢失啊？

面试官问你这个问题，其实是个开放题，他就是看看你有没有从架构角度整体构思和设计的思维以及能力。确实这个问题可以刷掉一大批人，因为大部分人平时不思考这些东西。

# 第八章电商项目篇之谷粒商城

## 一、常见共性问题

## 1、介绍下最近做的项目

可以从2个方向出发

介绍项目背景、项目功能和自己负责的功能模块

介绍项目背景、项目使用技术栈和自己负责的功能模块

### 1.1 项目背景：

可以介绍项目是什么类型的（B2C、B2B2C、O2O这类），为什么要做这个项目，

有工作经验的找工作一般这样介绍：项目是自己公司开发，自己运营的，然后不断加功能进行迭代和维护；或者是项目定制的，给甲方客户开发的一个项目，上线后不负责维护和迭代，这样避免了很多后期问题，这两种都可以。

咱们可以借鉴以上的方式去介绍，刨除公司情况，以学习为主。

### 1.2 项目功能：

结合项目，进行主要的功能模块阐述，可以结合电商项目的核心购物流程去说：后台管理系统（商品的管理）、商品详情、商品搜索、购物车、单点登录+社交登录、订单、支付、秒杀等等。

### 1.3 技术栈：

使用springboot整合SpringCloud 以及MyBatis-Plus进行微服务构建，使用nacos作为注册中心和配置中心，使用feign进行服务远程调用，使用gateway网关进行请求负载、请求过滤、统一鉴权和限流，使用Sentinel进行服务的熔断和降级，使用Spring Cloud Sleuth进行链路追踪，针对于项目图片文件资源较多，采用FastDFS进行文件资源存储，使用redis数据库进行数据缓存以及分布式锁的实现，使用ElasticSearch进行商品的搜索业务实现…..（这块基础架构说完后，主要结合自己负责的功能模块去说技术点的应用）

### 1.4 自己负责的功能模块：

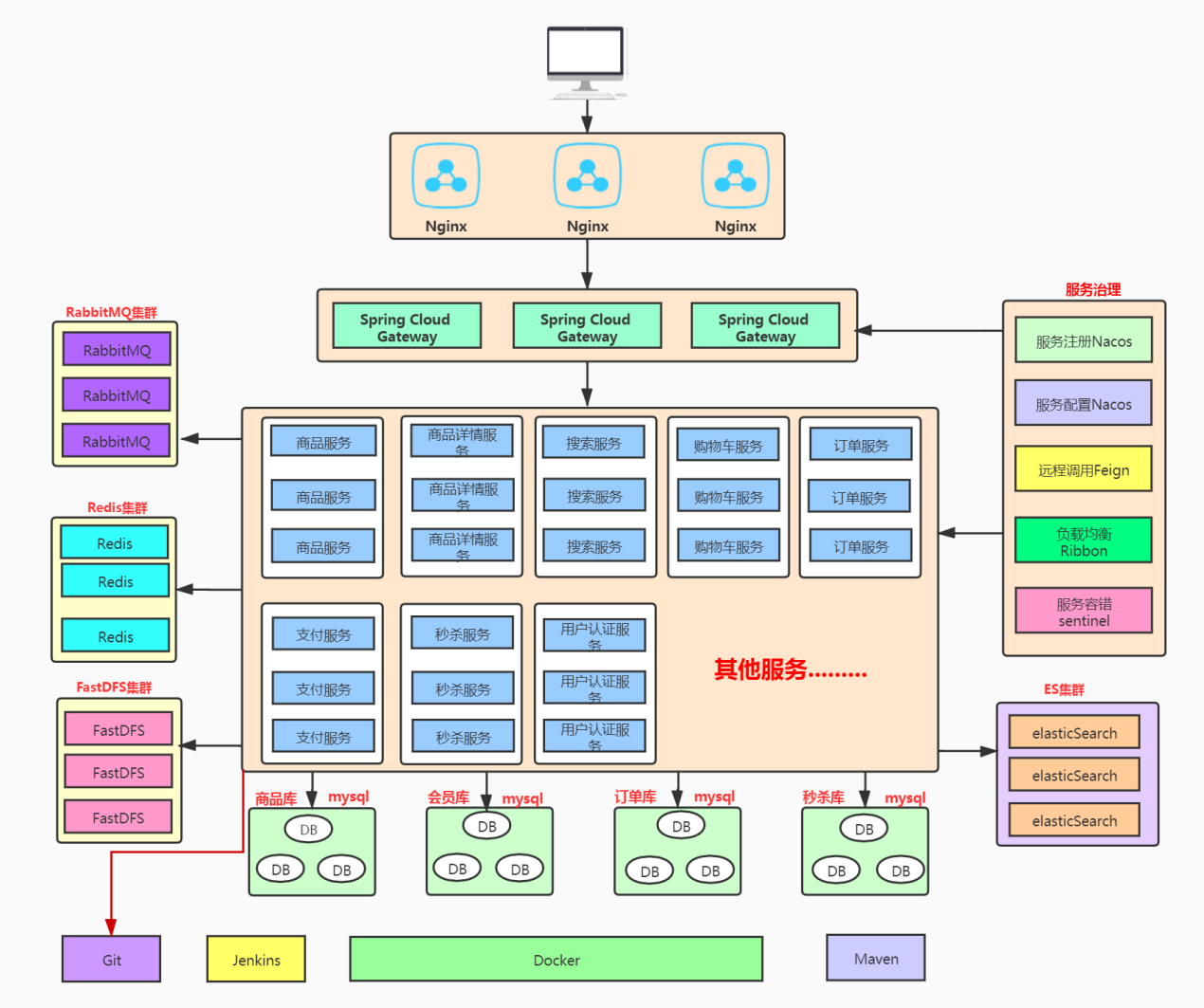
已简历为主，简历上写了哪几个，就说那几个，一定要知道自己简历写的什么内容。

### 1.5 项目介绍参考：

谷粒商城是B2C模式的综合性在线销售平台。商城分为后台管理部分与用户前台使用部分。后台管理部分包括：商品管理模块（商品分类、品牌、平台属性、SPU与SKU以及销售属性、商品上下架和商品评论管理等）、内容广告模块、库存管理模块、订单管理模块、促销管理（秒杀等商品设置）、客户模块、统计报表模块和系统基础权限等模块。

用户前台使用部分：商城首页、商品搜索（可按条件查询展示）、商品详情信息展示、购物车、用户单点登录和社交登录（微信登录）、用户会员中心、订单的创建修改、展示以及在线支付（支付宝、微信）、物流模块、商品评论以及秒杀活动等功能。

### 1.6 项目架构图：



### 1.7 整体业务介绍：

|  |  |
| --- | --- |
| 首页 | 静态页面，包含了商品分类，搜索栏，商品广告位。 |
| 全文搜索 | 通过搜索栏填入的关键字进行搜索，并列表展示 |
| 分类查询 | 根据首页的商品类目进行查询 |
| 商品详情 | 商品的详细信息展示 |
| 购物车 | 将有购买意向的商品临时存放的地方 |
| 单点登录 | 用户统一登录的管理 |
| 结算 | 将购物车中勾选的商品初始化成要填写的订单 |
| 下单 | 填好的订单提交 |
| 支付服务 | 下单后，用户点击支付，负责对接第三方支付系统。 |
| 订单服务 | 负责确认订单是否付款成功，并对接仓储物流系统。 |
| 仓储物流 | 独立的管理系统，负责商品的库存。 |
| 后台管理 | 主要维护类目、商品、库存单元、广告位等信息。 |
| 秒杀 | 秒杀抢购完整方案 |

### 1.8 后台管理系统功能：

电子商务网站整个系统的后端管理，按功能划分为九大模块，包括商品组织管理、订单处理、内容发布管理等模块。

#### 1.8.1 后台主页：

各类主要信息的概要统计，包括客户信息、 订单信息、商品信息、库存信息、评论和最近反馈等。

#### 1.8.2 商品模块：

##### 1).商品管理：

商品SPU和SKU的添加、修改、 删除、复制、批处理、商品计划上下架、SEO、商品多媒体上传等，可以定义商品是实体还是虚拟，可以定义是否预订、是否缺货销售等。

##### 2).商品分类管理：

树形的商品目录组织管理，并可以设置品类关联与商品推荐。

##### 3).商品平台属性管理：

定义商品的属性类型，设置自定义属性项。

##### 4).品牌管理：

添加、修改、删除、上传品牌 LOGO。

##### 5).商品评论管理：

商品评论的搜索、条件查询列表展示、回复、删除等功能。

#### 1.8.3 销售模块：

##### 1).促销秒杀管理：

设置秒杀商品、购物车促销和 优惠券促销三类，可以随意定义不同的促销规则，满足日常促销活动：购物折扣、购物赠送积分、购物赠送优惠券、购物免运输费、特价商品、特定会员购买特定商品、折上折、买二送一等。

##### 2).礼券、积分管理：

比如添加、发送礼券和积分

##### 3).关联/推荐管理：

基于规则引擎，可以支持多种推荐类型，可手工添加或者自动评估商品。

#### 1.8.4 订单模块：

##### 1).订单管理：

可以编辑、解锁、取消订单、 拆分订单、添加商品、移除商品、确认可备货等，也可对因促销规则发生变化引起的价格变化进行调整。订单处理完可发起退货、换货流程。

##### 2).支付：

常用于订单支付信息的查看和手工 支付两种功能。手工支付订单，常用于“款到发货”类型的订单，可理解为对款到发货这类订单的一种补登行为。

##### 3).结算：

提供商家与第三方物流公司的结算 功能，通常是月结。同时，结算功能也是常用来对“货到付款”这一类型订单支付后的数据进行对帐

#### 1.8.5 库存模块：

##### 1).库存管理：

引入库存的概念，不包括销售规则为永远可售的商品，一个SKU对应一个库存量。库存管理提供增加、减少等调整库存量的功能;另外，也可对具具体的SKU设置商品的保留数量、最小库存量、再进货数量。每条SKU商品的具体库存操作都会记录在库存明细记录里边。

##### 2).查看库存明细记录。

##### 3).备货/发货：

创建备货单、打印备货单、打印发货单、打印快递单、完成发货等一系列物流配送的操作。

##### 4).退/换货：

对退/换货的订单进行收货流程的处理。

#### 1.8.6 内容模块：

##### 1).内容管理：

包括内容管理以及内容目录管理。内容目录由树形结构组织管理。类似于商品目录的树形结构，可设置目录是否为链接目录。

##### 2).广告管理：

添加、修改、删除、上传广告、 定义广告有效时限。

##### 3).可自由设置商城导航栏目以及栏目内容、栏目链接。

#### 1.8.7 客户模块：

##### 1).客户管理：

添加、删除、修改、重设密码、 发送邮件等。  

##### 2).反馈管理：

删除、回复。

##### 3).消息订阅管理：

添加、删除、修改消息组 和消息、分配消息组、查看订阅人。

##### 4).会员资格：

添加、删除、修改。

#### 1.8.8 系统模块：

##### 1).安全管理：

管理员、角色权限分配和安全日志

##### 2).系统属性管理：

用于管理自定义属性。可关联模块包括商品管理、商品目录管理、内容管理、客户管理。

##### 3).运输与区域：

运输公司、运输方式、运输 地区。

##### 4).支付管理：

支付方式、支付历史。

##### 5).包装管理：

添加、修改、删除。

##### 6).数据导入管理：

商品目录导入、商品导入、 会员资料导入。

#### 1.8.9 报表模块：

   缺省数个统计报表，支持时间段过滤、支持按不同状态过滤、支持HTML、PDF和Excel格式的导出和打印。

1.用户注册统计   2.低库存汇总   3.缺货订单   4.订单汇总   5.退换货

## 2、项目开发周期：

**开发、维护和运营一体化的：**开发周期8个月左右，后期维护与迭代时间会更长

**项目定制：**前期架构+数据库设计+编码+开发+测试解bug共7个月左右进行项目交付

**培训学习项目:** 20天教程（咱们是学习20天课程）

## 3、项目参与人数：

**一般公司:**项目经理（PM）1人、产品（PD）2人、界面设计（UI）2人、前端 3人、Java后台（DE）6人，其中1人是开发组长、测试（QA）2人、运维（SRE）1人

**培训学习项目：**根据课程内容编写代码，自己实现部分功能

## 4、公司开发相关各岗位职责：

### 4.1 项目经理（PM）：

企业建立以项目经理责任制为核心，对项目实行质量、安全、进度、成本管理的责任保证体系和全面提高项目管理水平设立的重要管理岗位。职责：

1、负责软件项目管理及计划实施；

2、具备较强管理、协调及沟通能力，帮助开发人员解决开发过程中遇到的技术问题，做好日常的开发团队管理工作；

3、与各团队协同工作，确保开发工作正常顺利的开展；

### 4.2 产品（PD）：

企业中专门负责产品管理的职位，负责调查并根据用户的需求，确定开发何种产品，选择何种技术、商业模式等。并推动相应产品的开发组织，他还要根据产品的生命周期，协调研发、营销、运营等，确定和组织实施相应的产品策略，以及其他一系列相关的产品管理活动。职责： 1. 根据公司产品及用户需求，结合市场调研情况，进行产品规划；

2. 负责用户沟通、需求分析诊断；

3. 负责产品定位、用户体验流程定位及产品设计；

4. 推动、协调与控制产品策划及研发工作，保证产品需求的有效实现；

5. 负责产品持续升级，不断提升用户满意度及忠诚度；

6. 对行业及竞争产品的分析，跟踪最新发展趋势，并提交分析报告。

### 4.3 界面设计（UI）：

对软件的人机交互、操作逻辑、界面美观的整体设计。职责：

1、负责公司产品PC端和移动端的UI界面设计工作；

2、配合完成校样修改和界面调整；

3、深入了解负责的产品，并通过各种设计形式和视觉语言让用户感受到产品的优点和特性；

4、跟进设计的变化和需求，注重相关文档的整理、资料的收集；能独立完成界面设计工作。

### 4.4 开发组长（TL）：

其实就是个更小一点的项目经理。其职责：1、 参与软件的设计负责系统需求的分析，进行系统设计和数据库设计；

2、 解决开发过程中技术问题和提供解决办法；

3、 能够带领小组负责模块的功能开发；

4、 负责项目组代码的审查工作，有效地控制项目的质量风险。

### 4.5 测试（QA）：

测试工程师，软件质量的把关者，工作起点高，发展空间大。职责： 1.理解、分析需求文档，挖掘、细化需求;

2.根据软件需求及设计文档编写测试用例，参与文档评审并维护相关文档;

3.准备测试数据，执行测试用例，记录测试结果，整理测试报告;

4.负责BUG的提交、跟踪、验证、关闭;

5.负责测试部门测试环境及BUG系统管理与维护。

6.对产品进行必要的功能，性能，安全，兼容性及其它方面的测试工作；

7.公司安排的其它工作。

### 4.5 运维（SRE）：

运维工程师最基本的职责都是负责服务的稳定性。

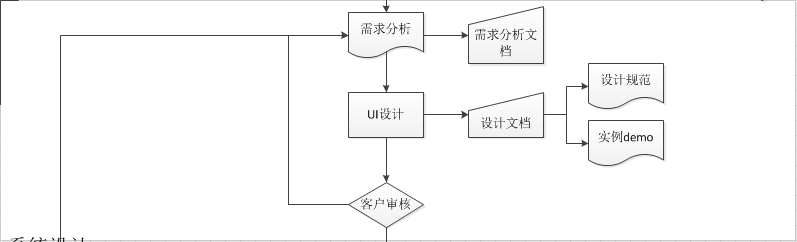
1. 产品发布前：负责参与并审核架构设计的合理性和可运维性，以确保在产品发布之后能高效稳定的运行。

2. 产品发布阶段：负责用自动化的技术或者平台确保产品可以高效的发布上线，之后可以快速稳定迭代。

3. 产品运行维护阶段：负责保障产品7\*24H稳定运行，在此期间对出现的各种问题可以快速定位并解决；在日常工作中不断优化系统架构和部署的合理性，以提升系统服务的稳定性。

## 5、企业项目开发流程:

### 3.1 需求分析



项目前期主要指的是项目业务需求调研、包括配合用户制定项目建设方案、技术规范书、配合市场人员进行售前技术交流等环节，此阶段应该组织由售前工程师、需求分析师以及系项目经理等组成一个临时小组，负责跟踪项目。这个小组根据项目的大小和客户的要求确定小组成员。

项目前期小组的工作是项目的开始，这个小组工作成绩的优劣、工作质量的高低，将直接影响项目的成败。因此，从管理层的角度，一定要重视这个环节。

项目前期小组需要完成的工作包括以下方面：

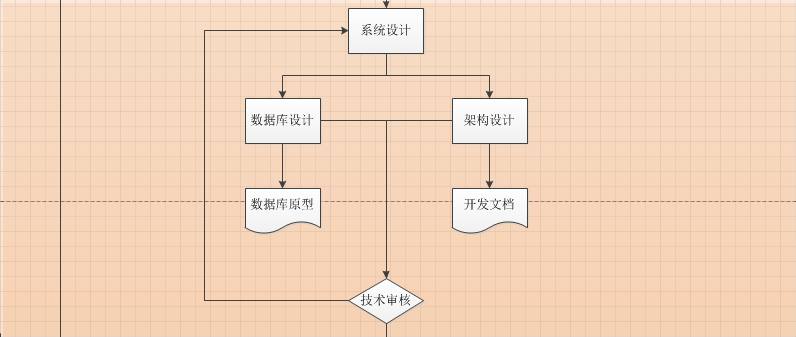
1、 客户的各种项目前期要求，如：方案介绍、业务需求编写等

2、 提交项目可行性分析报告，包括成本/效益分析

3、 提交项目建议方案

4、 提交业务需求说明书或需求分析说明书

### 3.2 系统设计



系统设计是决定项目或软件系统“怎样做”的过程，这个过程回答了系统应该如何实现的问题。从软件工程的角度，设计阶段大约是整个项目开发成本的25%，所以，设计团队以及该团队的工作成绩对于整个系统来说至关重要。

人员需求

设计团队一般由3—8名设计人员组成，从这个阶段起，项目需要一名项目经理，行使项目组的各种管理职能。设计团队的成员具体包括：

1名项目经理

包括1—2名项目前期成员

1名系统构架师

1名数据库设计人员

1名用户界面设计人员组成

设计团队需要完成的工作包括：

1、项目开发计划

2、确定系统软硬件配置最佳方案

3、确定系统开发平台以及开发工具

4、确定系统软件结构

5、确定系统功能模块以及各个模块之间的关系

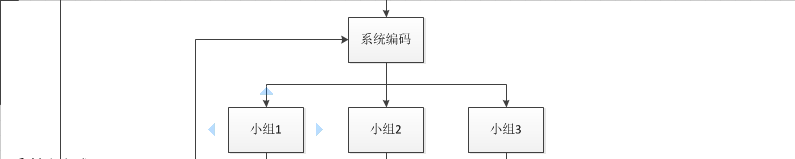
6、确定系统测试方案

7、提交系统数据库设计方案

8、提交系统概要设计文档

由于应用软件需求经常变化，因此设计需要考虑系统可扩展性，并需要在设计过程中对于重要的环节和用户进行及时沟通。

### 3.3 编码开发



将用户的需求变成真正可用的软件系统，是通过编码和系统实现阶段来完成的。虽然软件的质量主要取决于系统设计的质量，但是编码的途径和实现的具体方法对程序的可靠性、可读性、可测试性和可维护性产生深远的影响。

人员需求

这个阶段要根据用户对项目进度的要求灵活组织开发团队。为了工作的连贯性，同时也为了解决在开发过程中用户需求有可能变化的因素，开发团队应该保留1—3名设计团队的成员。

人员分工

开发过程中，项目经理的角色非常重要，项目经理负责项目组开发人员的日常管理，控制项目的进度，负责和设计部门、市场部门以及客户之间进行必要的沟通。这个阶段通常是多个部门的人员共同组成一个项目组，因此，项目管理的一定要保证统一管理，理想状态是项目经理全权负责项目组人员的人员工作安排、业绩考核、工资奖金等，因为项目经理最了解项目组成员的工作态度和工作业绩。

一般在大型项目开发团队中，应该设立专门的技术经理岗位，负责对项目组的技术方案进行管控，技术经理最好是由设计团队中抽调出来。技术经理在项目开发过程中需要注意程序风格、编码规范等问题，并必须进行有效的代码管理（版本管理）。

开发过程还应该进行系统的单元测试工作，确保各个独立模块功能的正确性和性能满足需求说明书的要求。

开发团队应该完成的工作包括：

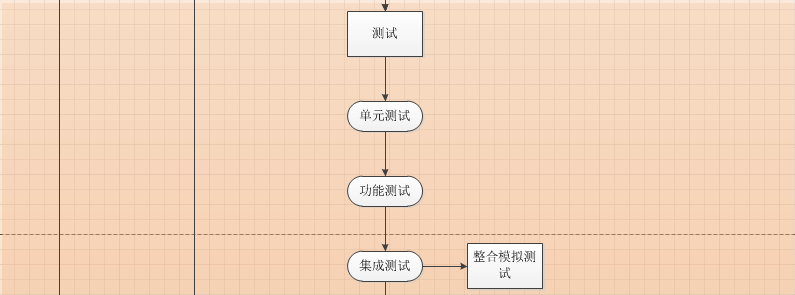
1、 系统的实现代码编写

2、 单元测试

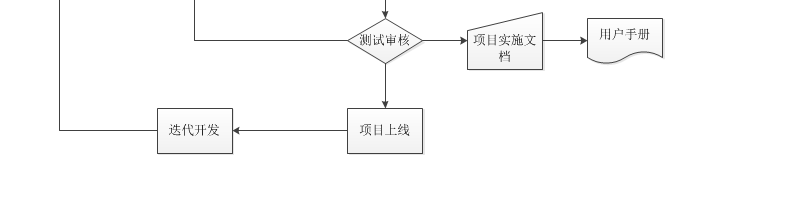
3、 提交源代码清单

4、 提交单元测试报告

### 3.4 系统测试



### 3.5 部署实施



由于从事的应用软件的开发，因此，在开发完成之后经常会有系统集成、软件的安装等工作。这个阶段还经常伴随着新的业务需求和本地化需求的产生，因此将会有一部分的开发工作需要在这个阶段完成。

## 6、项目版本控制：

使用git进行版本控制，剩下的主要是对git仓库的一些列操作，要记住操作命令，以及分支切换等等。

## 7、一般项目服务器数量：

### 开发测试阶段：

开发在自己电脑上开发，代码提交到git仓库（gitlab、gitee等）的开发分支，到测试时，公司有1-2台测试服务器，把开发分支代码合并到测试分支，使用jenkins进行测试环境代码部署，测试人员进行测试。

### 生产环境：

为了保证高可用，首先每个服务都要进行 集群部署，包括项目中依赖的第三方服务，这样的话，服务器的数量是很庞大的，以尚品汇商城所学功能实现为例，如下：

Nginx2台（主备）；Nacos 3台（官方推荐最少3台）；项目中有服务数量\*2（19\*2=38）共需要38台；redis无中心化集群6台（3主3从），如果是主从哨兵集群5台；mysql数据库（读写分离的情况下，1主2从 4\*3 =12）共12台；ES集群3台；rabbitmq集群2台；fastDFS集群（保证高可用情况下，traker2台，storage2组4台）共6台。

以上所述 服务器数量一共为72台，数量很多，以目前市场硬件服务器平均一台5万块钱来计算，72\*5=360万，投入成本和维护成本很大，在项目前期没有大用户量的情况下，不建议。

针对于这个问题，如果是项目定制（外包）的话，可以由甲方（客户）自己去部署，这种情况也是存在的。

还有一种情况是在项目前期没有那么大的用户量情况下，可以采用单机（主备）部署方案，把所有服务部署同一台服务器上，进行资源节省。

## 8、上线后QPS，用户量、同时在线人数并发数等问题：

建议采用项目定制方案：项目给客户做的，甲方的数据我们拿不到。

但是要了解以下数据关键词：

（1）QPS（TPS）：每秒钟request/事务 数量

（2）并发数： 系统同时处理的request/事务数

（3）响应时间： 一般取平均响应时间

QPS（TPS）= 并发数/平均响应时间或者并发数 = QPS\*平均响应时间 一个典型的上班签到系统，早上8点上班，7点半到8点的30分钟的时间里用户会登录签到系统进行签到。公司员工为1000人，平均每个员上登录签到系统的时长为5分钟。可以用下面的方法计算。 QPS = 1000/(30\*60) 事务/秒 平均响应时间为 = 5\*60 秒 并发数= QPS\*平均响应时间 = 1000/(30\*60) \*(5\*60)=166.7

说明：

一个系统吞吐量通常由QPS（TPS）、并发数两个因素决定，每套系统这两个值都有一个相对极限值，在应用场景访问压力下，只要某一项达到系统最高值，系统的吞吐量就上不去了，如果压力继续增大，系统的吞吐量反而会下降，原因是系统超负荷工作，上下文切换、内存等等其它消耗导致系统性能下降。

我们做项目要排计划，可以多人同时并发做多项任务，也可以一个人或者多个人串行工作，始终会有一条关键路径，这条路径就是项目的工期。系统一次调用的响应时间跟项目计划一样，也有一条关键路径，这个关键路径是就是系统影响时间； 关键路径是有CPU运算、IO、外部系统响应等等组成。

## 9、如何解决并发问题的？

尚品汇商城是微服务架构构建，集群部署，使用nginx做负载，数据库的读写分离，redis缓存，页面静态化等这些都是解决并发的手段。

## 10、你做完这个项目后有什么收获？

首先，在数据库方面，我现在是真正地体会到数据库的设计真的是一个程序或软件设计的重要和根基。因为数据库怎么设计，直接影响到一个程序或软件的功能的实现方法、性能和维护。由于我做的模块是要对数据库的数据进行计算和操作的，所以我对数据库的设计对程序的影响是深有体会，就是因为我们的数据库设计得不好，搞得我在对数据库中的数据进行获取和计算利润、总金时，非常困难，而且运行效率低，时间和空间的复杂也高，而且维护起来很困难，过了不久，即使自己有注释，但是也要认真地看自己的代码才能明白自己当初的想法和做法。加上师兄的解说，让我对数据库的重要的认识更深一层，数据库的设计真的是重中之重。

其次，就是分工的问题。虽然这次的项目我们没有在四人选出一个组长，但是，由于我跟其他人都比较熟，也有他们的号码，然后我就像一个小组长一样，也是我对他们进行了分工。俗话也说，分工合作，分好了工，才能合作。但是这次项目，我们的分工却非常糟糕，我们在分工之前分好了模块，每个 责哪些模块。本以为我们的分工是明确的，后来才发现，我们的分工是那么的一踏糊涂，一些功能上紧密相连的模块分给了两个人来完成，使两个人都感到迷惘，不知道自己要做什么，因为两个人做的东西差不多。我做的，他也在做，那我是否要继续做下去？总是有这样的疑问。从而导致了重复工作，浪费时间和精力，并打击了队员的激情，因为自己辛辛苦苦写的代码，最后可能没有派上用场。我也知道，没有一点经验的我犯这样的错是在所难免，我也不过多地怪责自己，吸取这次的教训就好。分工也是一门学问。 再者，就是命名规范的问题。可能我们以前都是自己一个人在写代码，写的代码都是给自己看的，所以我们都没有注意到这个问题。就像师兄说的那样，我们的代码看上去很上难看很不舒服，也不知道我们的变量是什么类型的，也不知道是要来做什么的。但是我觉得我们这一组人的代码都写得比较好看，每个人的代码都有注释和分隔，就是没有一个统一的规范，每个人都人自己的一个命名规则和习惯，也不能见名知义。还有就是没有定义好一些公共的部分，使每个人都有一个自己的“公共部分”，从而在拼起来时，第一件事，就是改名字。而这些都应该是在项目一开始，还没开始写代码时应该做的。 然后，我自己在计算时，竟然太大意算错了利润，这不能只一句我不小心就敷衍过去，也是我的责任，而且这也是我们的项目的核心部分，以后在做完一个模块后，一定要测试多次，不能过于随便地用一个数据测试一下，能成功就算了，要用可能出现的所有情况去测试程序，让所有的代码都有运行过一次，确认无误。

最后，也是我比较喜欢的东西，就是大家一起为了一个问题去讨论和去交流。因为我觉得，无论是谁，他能想的东西都是有限的，别人总会想到一些自己想不到的地方。跟他人讨论和交流能知道别人的想法、了解别人是怎样想一个问题的，对于同样的问题自己又是怎样想的，是别人的想法好，还是自己的想法好，好在什么地方。因为我发现问题的能力比较欠缺，所以我也总是喜欢别人问我问题，也喜欢跟别人去讨论一个问题，因为他们帮我发现了我自己没有发现的问题。在这次项目中，我跟植荣的讨论就最多了，很多时候都是不可开交的那种，不过我觉得他总是能够想到很多我想不到的东西，他想的东西也比我深入很多，虽然很多时候我们好像闹得很僵，但是我们还是很要好的! 嘻嘻！而且在以后的学习和做项目的过程中，我们遇到的问题可能会多很多，复杂很多，我们一个人也不能解决，或者是没有想法，但是懂得与他人讨论与交流就不怕这个问题，总有人的想法会给我们带来一片新天地。相信我能做得更好。 还有就是做项目时要抓准客户的要求，不要自以为是，自己觉得这样好，那样好就把客户的需求改变，项目就是项目，就要根据客户的要求来完成。

## 11、在做这个项目的时候你碰到了哪些问题？你是怎么解决的？

（1）开发SpringBoot接口出现客户端和服务端不同步，导致接口无法测试，产生的原因沟通不畅。

（2）订单提交时由于本地bug或者意外故障导致用户钱支付了但是订单不成功，采用对账方式来解决。

（3）上线的时候一定要把支付的假接口换成真接口。

（4）项目中用到了曾经没有用过的技术，解决方式：用自己的私人时间主动学习

（5）在开发过程中与测试人员产生一些问题，本地环境ok但是测试环境有问题，环境的问题产生的，浏览器环境差异，服务器之间的差异

（6）系统运行环境问题，有些问题是在开发环境下OK，但是到了测试环境就问题，比如说系统文件路径问题、导出报表中的中文问题（报表采用POI），需要在系统jdk中添加相应的中文字体才能解决；

## 二、介绍功能模块

这块面试官的问法不同，有的会让你从你负责的模块中，挑选一个你认为做的最好的或者有亮点的模块去说，有的面试官是把你负责的模块问个遍，所以只要你简历上写了是你负责的模块，一定要去弄的非常熟练，如果你自己做的东西，你都不知道，那对于整个项目来说也就没必要再问了，所以这块是重点。在介绍自己负责的功能模块时，可以通过2个方向去介绍，一个是模块业务，另一个是模块开发中的技术解决方案。这两个方向全掌握了，才是真正的把负责的模块掌握了。

下面是所学项目中所有功能模块的业务以及常见面试问题，仅供参考，不要去背，要结合项目进行理解为主。

## 1、商品管理模块

数据库表设计：（重点，要知道表中字段与表直接的关系）



### 1.1 业务参考话术：

我们这个电商平台是不支持商家入驻功能的，因此整个商品管理主要是我们这个电商后台系统的一个核心功能模块。主要的功能分为：

对于商品品牌、商品分类、销售属性和平台属性进行管理的一些操作；

对商品有对应操作权限的业务员对于商品的添加、上下架、修改、删除、批量操作、模糊查询等功能。

针对商品这个模块来说，因为不同的分类对应不同的平台属性，平台属性和SKU进行关联，用于后期商品搜索功能实现，可以通过平台属性查询出所选属性下的所有SKU信息；

同时，一个分类下又有多个品牌，品牌下对应多个SPU商品，每一个SPU下有多种销售属性，我们通过商品SPU的不同销售属性组合生成多个SKU。例如 iPhone7是一个SPU，由于iPhone7分内存、颜色等销售属性，内存有64G、128G和256G，颜色有红色、金色和黑色，这样每种不同销售属性组合就是一个新的SKU，64G+黑色、64G+金色、64G+红色，128G+金色…….等。（主要就是说表设计和表关系）

### 1.2 商品添加

先判断是否有对应的商品添加的权限，如果没有，直接不显示操作按钮；

如果有，先选择商品所属的分类，然后商品分类选择之后会对应的查询出所有平台属性和所关联的所有品牌；

然后添加产品SPU的一些基本信息（SPU名称、标题、商城价格等）；

其次，上传产品SPU对应的图片，这块因为整个电商平台中需要保存大量的图片，因此我们使用了一个分布式文件存储系统FastDFS来存储商品的图片，这样的话后期也可针对容量进行水平扩展，不影响原来的使用，并且针对于高并发、高可用的问题，FastDFS的容灾性、负载均衡也是个优势；

在SPU列表中有添加sku的按钮，点击后可以添加给SPU添加SKU。进入SKU添加页面时会查出当前SPU的所有销售属性和图片，然后添加SKU信息，选择对应的销售属性，选择对应的图片。调用后台的商品添加接口/服务，然后对应的接口/服务中生成商品添加时的一些默认数据（添加时间、更新时间、操作人等），在商品添加的时候我们默认是下架状态。需要拥有审核功能的业务员针对商品中是否有一些不合法的数据进行审核。

### 1.3 商品上下架

因为商品添加之后，默认是下架状态，如果商品想处于销售状态，必须将商品的状态改为上架状态。在SKU列表页面中有上、下架操作按钮：

上架时将SKU信息添加Redis进行缓存，将SKU信息添加到ES一份以供搜索

修改上架状态 0未上架 1已上架 2已下架

### 1.4 模块相关问题：

#### 1）SPU与SKU的概念，SKU是怎么划分的？

比如，咱们购买一台iPhoneX手机，iPhoneX手机就是一个SPU，但是你购买的时候，不可能是以iPhoneX手机为单位买的，商家也不可能以iPhoneX为单位记录库存。必须要以什么颜色什么版本的iPhoneX为单位。比如，你购买的是一台银色、128G内存的、支持联通网络的iPhoneX ，商家也会以这个单位来记录库存数。那这个更细致的单位就叫库存单元（SKU）。

## 2、商品详情模块

### 2.1 业务参考话术：

商品详情页，就是以购物者的角度展现一个sku的详情信息。这个页面不同于传统的页面，使用者并不是管理员，需要对信息进行查删改查，取而代之的是点击购买、放入购物车、切换颜色等等。另外一个特点就是该页面的高访问量，虽然只是一个查询操作，但是由于频繁的访问所以我们必须对其性能进行最大程度的优化。所以我们页面数据放入Redis缓存来提高性能，为了防止缓存击穿，我们采用了分布式锁。由于商品详情展示的数据需要要调用多个查询接口，为了提高响应速度，我们采用线程异步编排的方式进行接口调用。

### 2.2 模块相关问题：

Redis是这块的重点之一，有关reids的高频问题，详见高频面试题，这里只说项目相关redis常见问题

#### 1）Redis在你们项目中是怎么用的？

商品详情中的数据放入缓存，价格是实时获取的；

单点登录系统中也用到了redis。因为我们是微服务系统，把用户信息存到redis中便于多系统之间获取共用数据；

我们项目中同时也将购物车的信息设计存储在redis中，用户未登录采用UUID作为Key，value是购物车对象；用户登录之后将商品添加到购物车后存储到redis中，key是用户id，value是购物车对象；

因为针对评论这块，我们需要一个商品对应多个用户评论，并且按照时间顺序显示评论，为了提高查询效率，因此我们选择了redis的list类型将商品评论放在缓存中；

在统计模块中，我们有个功能是做商品销售的排行榜，因此选择redis的zset结构来实现；

#### 2）缓存击穿解决（分布式锁）

缓存击穿是指对于一些设置了过期时间的key，如果这些key可能会在某些时间点被超高并发地访问，是一种非常“热点”的数据。这个时候，需要考虑一个问题：如果这个key在大量请求同时进来之前正好失效，那么所有对这个key的数据查询都落到db，我们称为缓存击穿。

使用分布式锁，采用redis的KEY过期时间实现

Redis:命令

# set skuid:1:info “OK” NX PX 10000

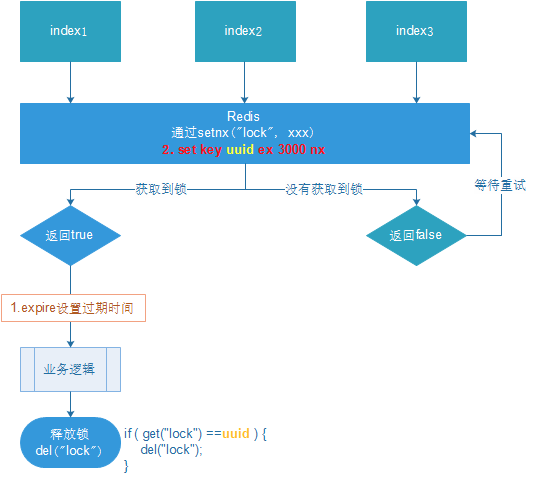
EX second ：设置键的过期时间为 second 秒。

PX millisecond ：设置键的过期时间为 millisecond 毫秒。

NX ：只在键(key)不存在时，才对键(key)进行设置操作。

XX ：只在键(key)已经存在时，才对键(key)进行设置操作。

Redis SET命令用于设置给定key的值，如果key已经存在其他值，SET就会覆盖，且无视类型。



由于redis删除操作缺乏原子性，可以使用lua脚本进行删除，但是由于redis主从切换时，有误删锁的情况，所以不建议使用，后来咱们演变为redission框架进行分布式锁的实现，原理和redis的一样，redission给进行了封装，通过调用lock以及unlock方法进行加锁和解锁，使用更简单。

最终我们自定义了一个注解结合AOP+redission框架的方式，把这个分布式锁加以复用，需要用到分布式锁的地方可以直接打注解就可以了。

**有关于redis分布式锁的演变，你可以当做在开发中遇到的问题来讲。**

## 3. 商品搜索模块

### 3.1 业务参考话术：

商品搜索是根据关键字进行拼配，从已有的数据库中摘录出相关的记录反馈给用户，在我们项目中有两个搜索入口，一个是商品的三级分类还有个就是用户在搜索框中输入关键字。我们主要是采用ES来实现业务，在ES中我们主要存入了分类id与名称、skuId、价格、名称、图片、品牌的id、名称、logo的url、平台属性数据以及热度排名字段等内容，我们把商品名称使用ik分词器进行了中文分词，同时还可以通过平台属性值、分类、价格区间以及品牌等作为搜索过滤条件，给用户展示商品名称、价格和默认图片等信息。

### 3.2 模块相关问题：

#### 1）ES的倒排索引？

正排索引：根据id去找到相对应的词

|  |  |
| --- | --- |
| Id（索引） | 词 |
| 1 | 红海行动 |
| 2 | 红海事件 |
| 3 | 红海行动事件 |
| 4 | 湄公河行动 |

倒排索引：根据词，看这个词都在哪个id（索引）里出现了

|  |  |
| --- | --- |
| 词 | Id |
| 红海 行动 | 红海1、2、3；行动1、3、4 |
| 红海 事件 | 红海1、2、3；事件2、3 |
| 红海 行动 事件 | 红海1、2、3；行动1、3、4；事件2、3 |
| 湄公河 行动 | 湄公河4；行动1、3、4 |

#### 2）ES过滤条件有哪些？

分类信息、平台属性、品牌、价格区间、热度、评论数等

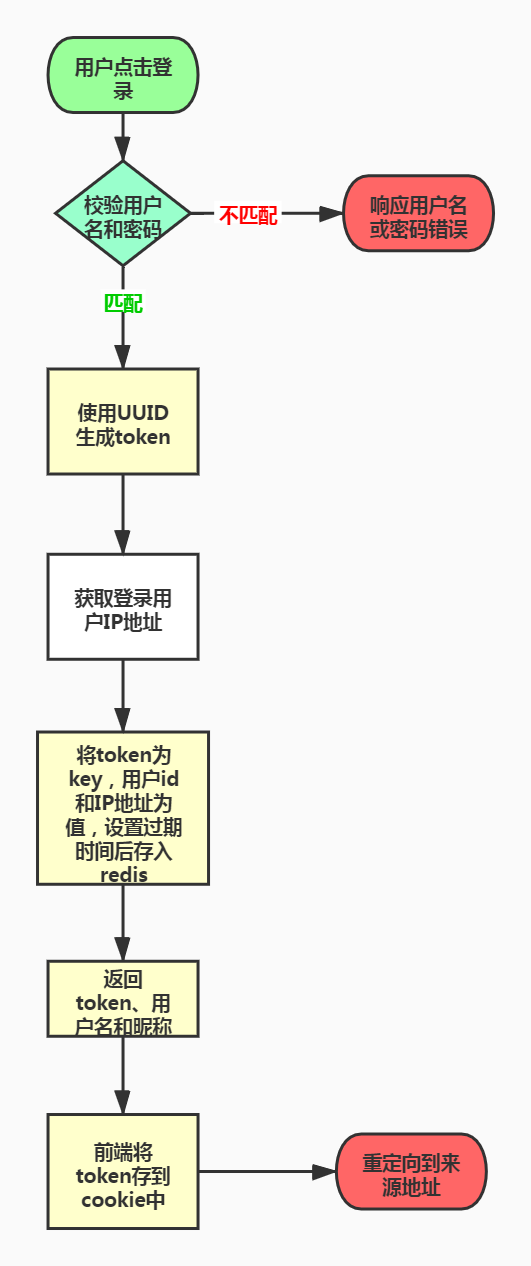
#### 3）ES数据同步问题怎么处理？

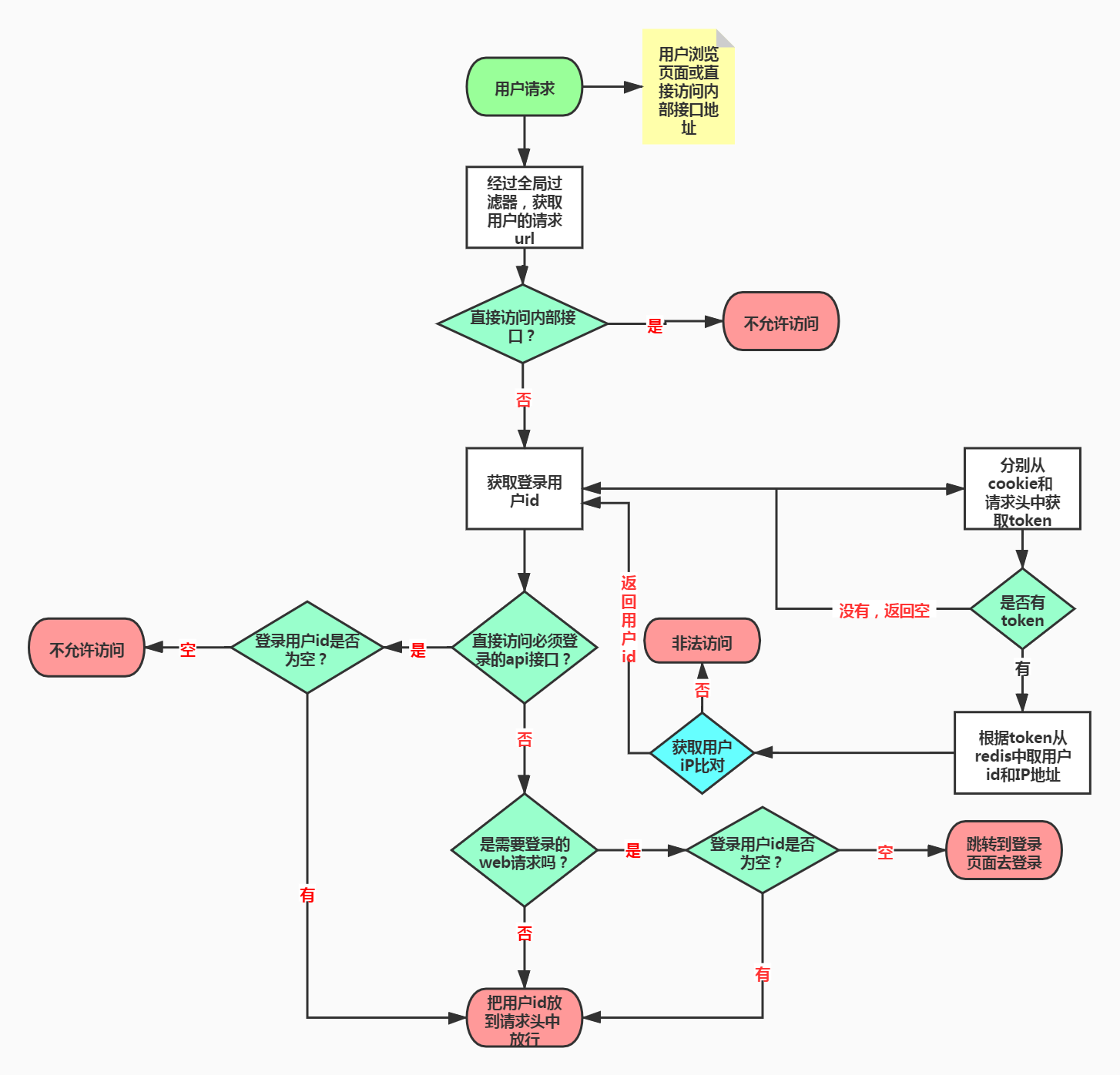
当商品上架时，发送mq消息通知搜索服务，当搜索服务监听到消息后把商品数据加入ES中，同样，当商品下架时也是通过mq进行消息传输来实现数据的最终一致。

## 4. 单点登录模块

### 4.1 业务参考话术：

单点登录就是访问项目时，用户只需要登录一次，就可以去访问所以内容。我们是采用网关全局过滤器、token认证以及redis来实现， 登录业务，用接收的用户名密码核对后台数据库，核对通过，用uuid生成token，将用户id加载到写入redis，redis的key为token，value为用户id。登录成功返回token与用户名和昵称，将token与用户信息记录到cookie里面重定向用户到之前的来源地址。用户再次访问其他需要登录的内容时通过网关过滤器进行拦截校验。具体业务参考以下流程图：





### 4.2 模块相关问题：

#### 1）cookie被禁用了能登录吗？怎么解决的？

不能登录，因为token存在cookie中，解决办法是参考京东的给用户提示：请您启用浏览器Cookie功能或更换浏览器

#### 2）怎么防止cookie盗用（盗链）？

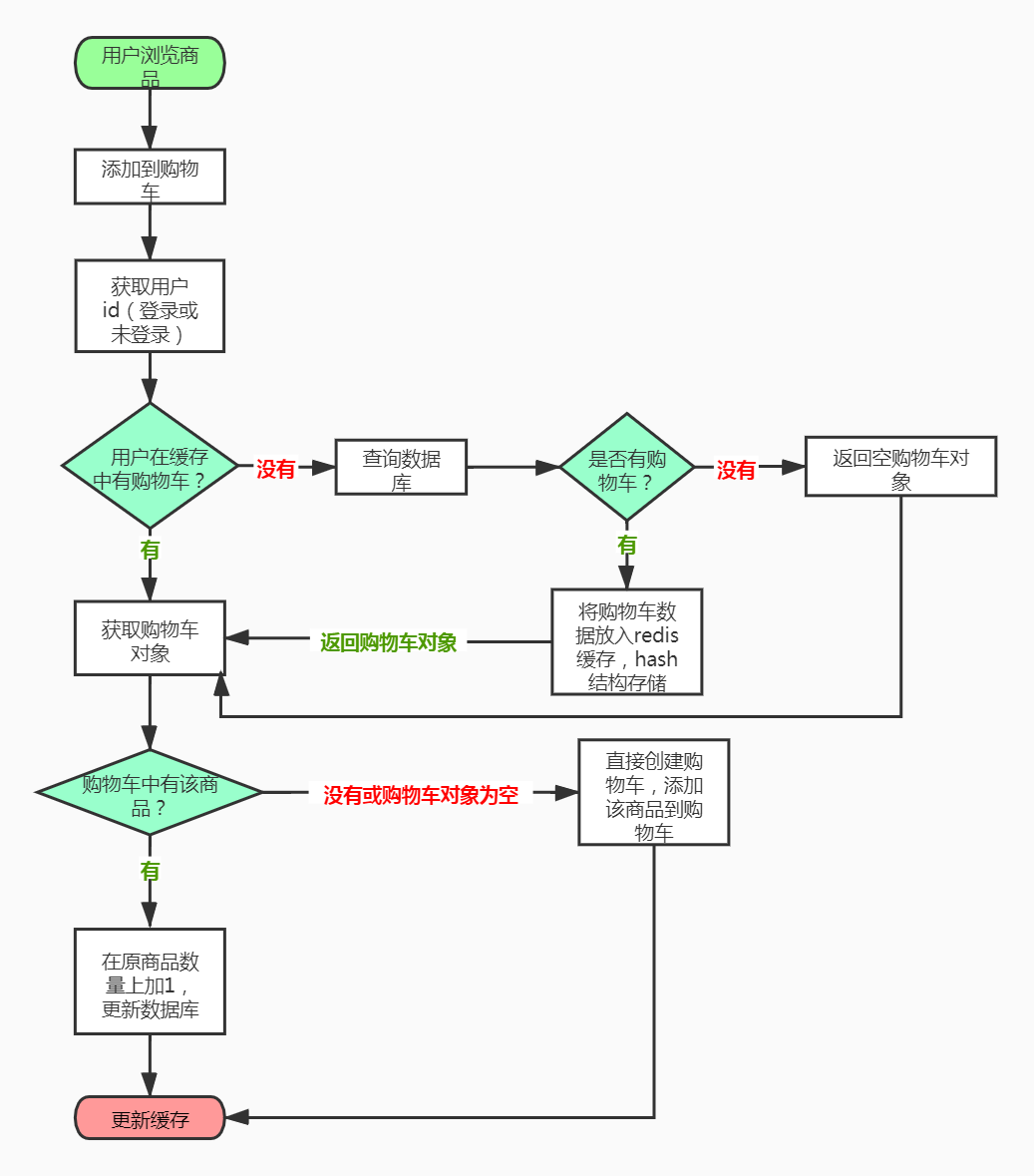
采用https进行加密传输，或者在token中加入以用户访问ip作为秘钥，解密token比对用户访问ip地址，ip不匹配，重新登录。

## 5. 购物车模块

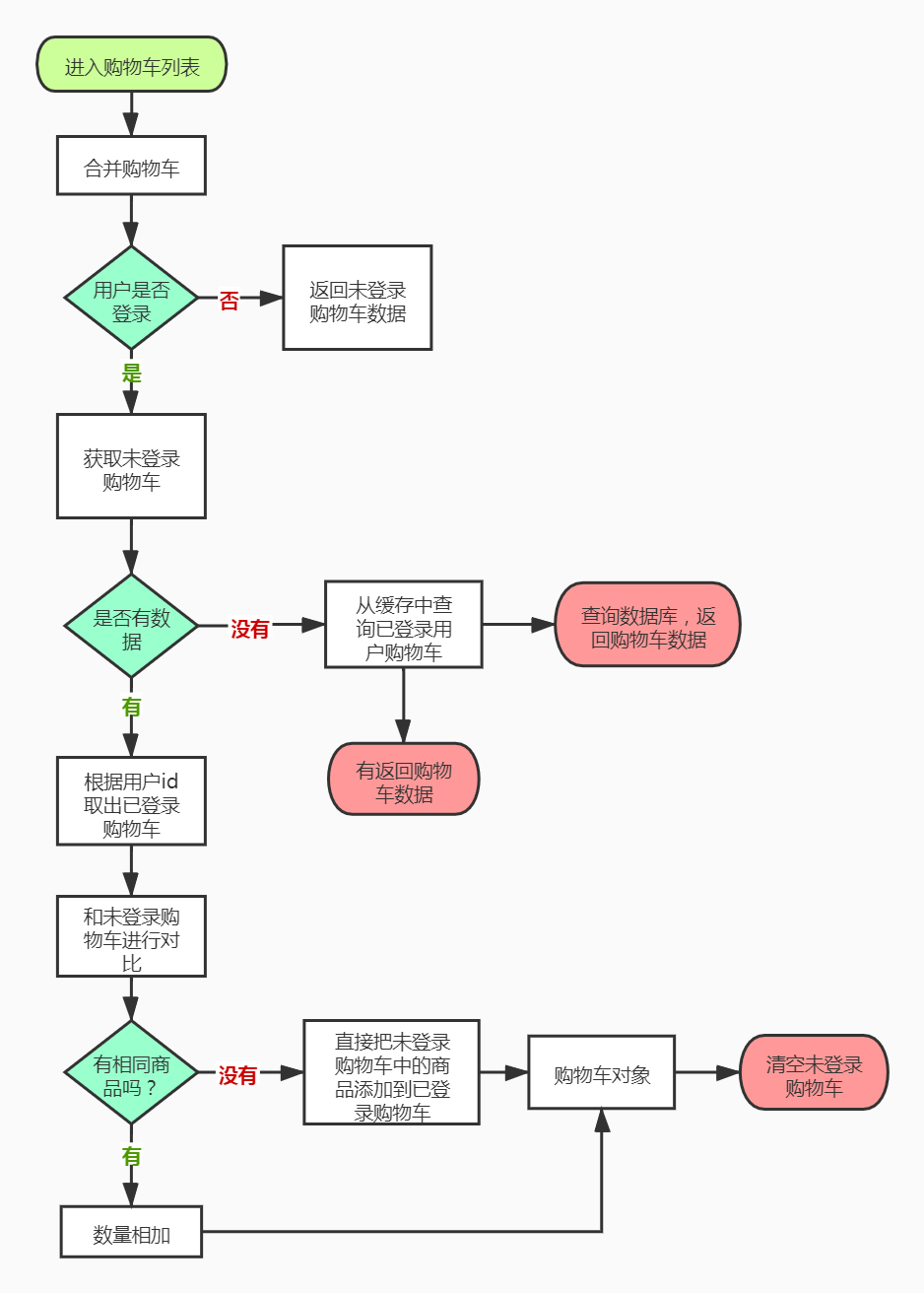
### 5.1 业务参考话术：

我们项目对于购物车的设计主要分为两个部分：用户未登录和已登录两种状态。在添加购物车时需要进行判断用户是否登录，如果未登录，则将购物车的信息放在redis中，采用hash结构存储，key为uuid（用户的临时id）随机生成，field为skuId，value是商品购物项信息，同时在数据库进行持久化操作，将reids的Key（临时用户id）存放到cookie中。用户已登录状态下，cookie中会存放token，根据token我们能获取到用户的id，所以采用userId作为hash结构的key，field为skuId，value是商品购物项信息。在进入购物列表时，会判断用户的登录状态，如果是未登录的话直接返回未登录购物车数据，如果是已登录，就判断是否存在未登录购物车，存在就进行购物车合并，然后再删除未登录购物车数据。

添加购物车流程：



合并购物车流程：



### 5.2 模块相关问题：

#### 1）购物车未登录有过期时间吗？已登录后有吗？

未登录的话：7天或15天

淘宝的设计是没有未登录购物车（如果面试不好说，咱们也可以说这种模式）

已登录情况下是永久存储，reids的数据有过期，主要是释放 redis’中数据，数据库采用永久存储。

#### 2）购物车能添加多少个商品？

49件或者99件商品，只要合理就行

#### 3）每个商品最多能加多少件？

库存充足的情况下可以添加99个，京东是200个

#### 4）添加购物车减库存吗？

添加购物车不减库存，验库存数量

#### 5）进入购物车列表的时候验价格吗？

验价格，如果价格有变化，展示新价格，给用户提示

#### 6）购物车里的商品价格变换了有提示吗？库存无货了有提示吗？

因为进入购物车列表时进行库存和价格的验证，所以会有提示。

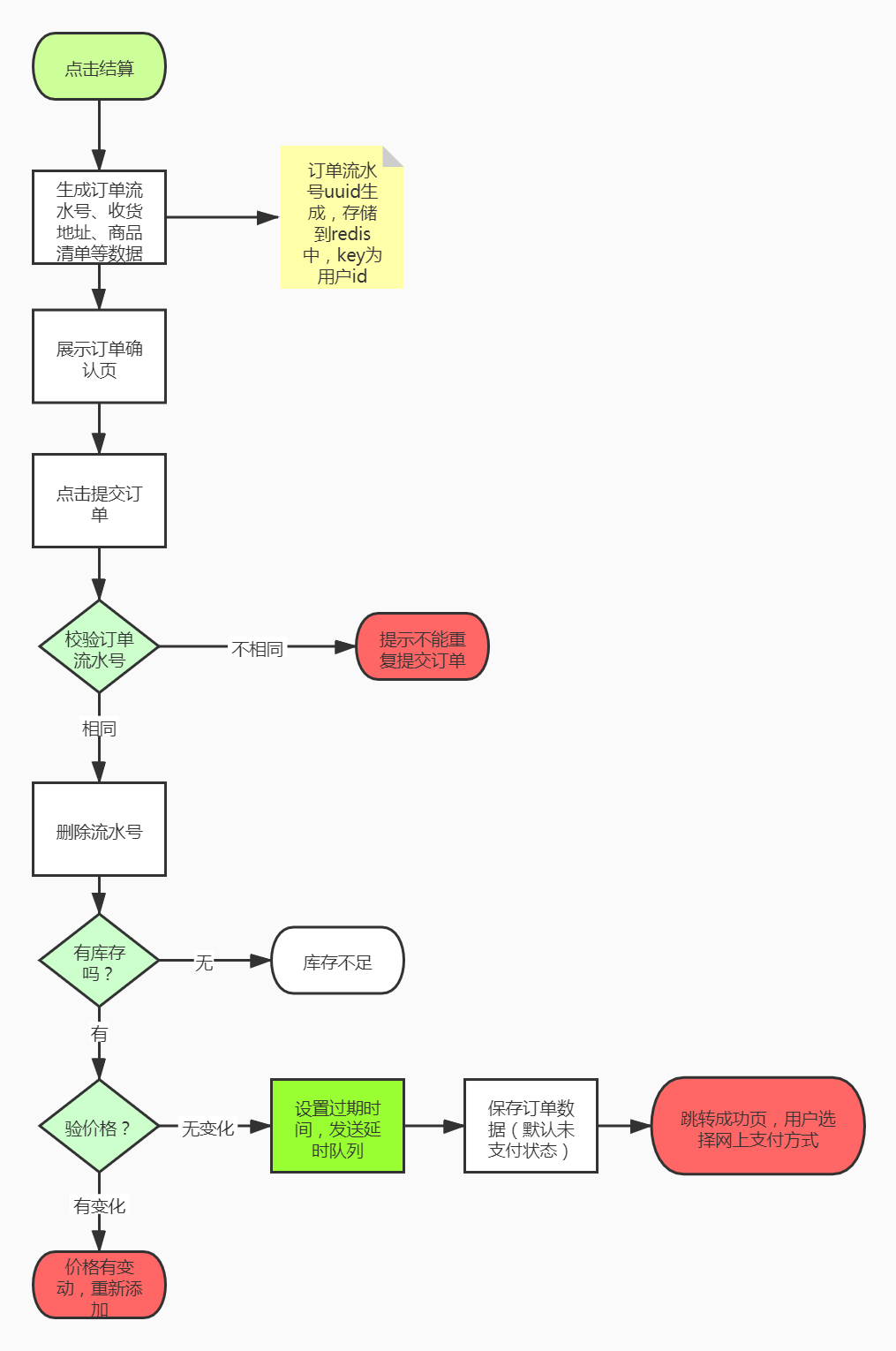
## 6. 订单微服务：

### 6.1 业务参考话术：

整个订单模块有结算页、下单、对接支付服务和对接库存管理系统等功能，当用户发起结算请求时，由于用户在没有登录的情况下也可以点击结算，所以我们要先判断用户是否登录，只有已登录的情况下，才能跳转到结算页面，在结算页，用户可以选择收货地址，给用户展示订单信息，用户选择支付方式，提供了微信支付和支付宝支付，确认订单信息，然后提交数据到后台，生成对应的订单表、订单详情表和订单物流表（当订单生成的时候，我们要调用对应的库存系统针对订单的商品数量进行验库存，还要进行验价格）。当订单创建成功之后，自动跳转到成功页面（将订单数据和到期时间传递过去）。

这块我们设置的订单的有效时间为24小时（这个时间可以自己定，只要合理就行），因为我们利用延时队列实现定时消息发送，消费者到时间后监听到消息，进行订单校验，如果订单是未支付状态，把订单状态修改为关闭订单。

订单的主要状态有 未支付、已支付、待发货、已收货、待评价、已完成、已关闭等。



### 6.2 模块相关问题：

#### 1）订单有效期是多久，怎么取消订单？

订单有效期为24小时，我们采用rabbitmq的延时队列，在生成订单时，发送延时消息，设置消息24小时后触发，然后监听到这个消息后，进行订单的取消。

#### 2）怎么防止订单重复提交？

在进入结算页面是我们生产了一个流水号，然后保存到结算页面的隐藏元素中一份，Redis中存一份，每次用户提交订单时都检查reids中流水号与页面提交的是否相符，如果相等可以提交，当订单保存以后把后台的流水号删除掉。那么第二次用户用同一个页面提交的话流水号就会匹配失败，无法重复保存订单。

#### 3）订单超卖问题怎么解决的？（库存只剩1件商品，多个用户同时下单）

我们采用的是下订单不减库存，只验证库存，在支付成功后再去扣减库存，如果库存扣减失败，通知后台进行补货，如果这个商品不能补货，人工客户介入，和买家进行沟通，给予退款或相应补偿。

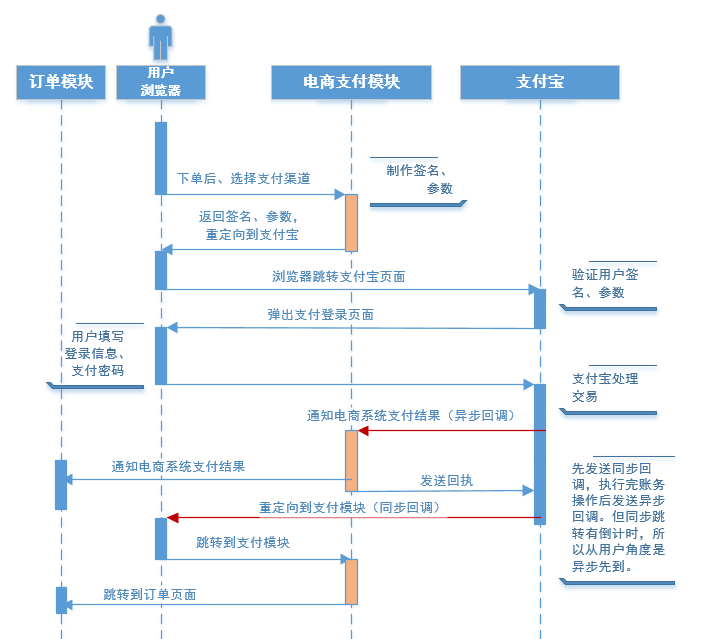
如果想实现不超卖，就得在下单时进行库存锁定，（可以使用数据库锁方式）然后减库存操作。

## 7. 支付微服务

### 7.1 业务参考话术：

当用户发起支付请求时，我们采用支付宝或者微信支付，以支付宝为例，我们要制作调用支付宝统一下单接口（支付接口）需要的签名和参数，我们先封装了一个AlipayClient配置类，把商户appid，支付宝公钥，开发者私钥，统一下单接口url以及签名方式等默认参数进行传入，从而生成AlipayClient对象交于spring管理。当调用支付时，我们会把订单交易标号、商品码，交易金额以及同步回调地址和异步回调地址等参数传给支付宝，从而生成支付页面，在这个过程中我们会记录支付信息存储到数据库中，当用户扫码支付后，支付宝回给用户跳转页面，主要是我们设定的同步回调地址。平台这块是根据支付宝的异步回调来判断用户是否支付成功，接收到异步回调时我们会验证签名，以校验合法性，同时会使用rabbitmq给订单模块发送一个校验成功的消息，以便于订单模块的后续处理。

支付宝还提供了退款、对账、关闭交易等接口，在项目中也都使用了。



### 7.2 模块相关问题：

#### 1）支付宝支付需要的参数？

主要参数有appid、开发者私钥、支付宝公钥、支付网关地址，同步回调url、异步回调url、请求数据格式、编码方式、签名方式、订单交易编号、订单金额、商品编码等

#### 2）支付锁库存吗？

我们做的是支付不锁库存，只有在支付成功后，去锁定库存进行响应的扣减。这块面试官会说超卖了，可以参考上面订单超卖内容进行解答。

#### 3）支付日志表（信息表）中记录了什么东西? 有什么作用？

主要记录 支付类型、金额、交易号、订单编号、交易内容、支付状态和时间等。

支付日志表主要是记录支付的状态转换和数据以便于和支付宝微信等三方平台进行对账等。

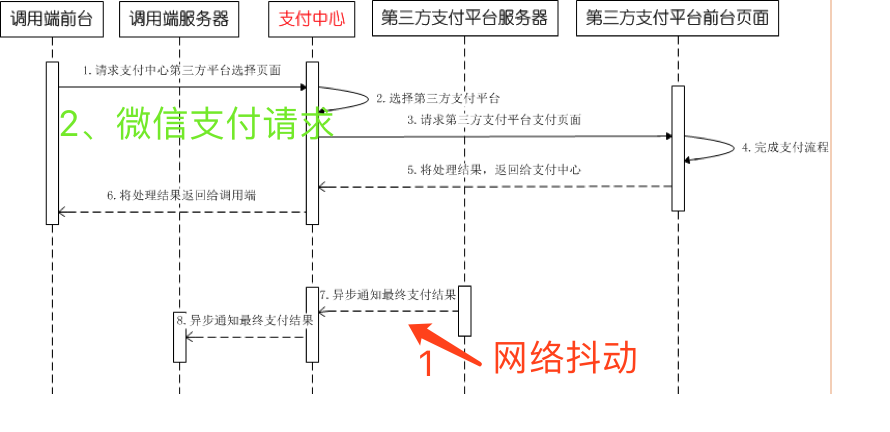
#### 4）重复支付问题？

##### 场景1

对同一个支付平台web端和app端的重复支付，同一个支付平台针对同一笔业务订单生成的流水号是唯一的，第三方支付平台对同一个流水号只进行一次支付，不会重复支付。

##### 场景2

例如银行的扣款成功的通知结果延迟了，在同一笔业务订单因为没有收到成功的结果通知，发生第二次支付并正常成功后，才收到第一次支付成功的通知。如下图所示

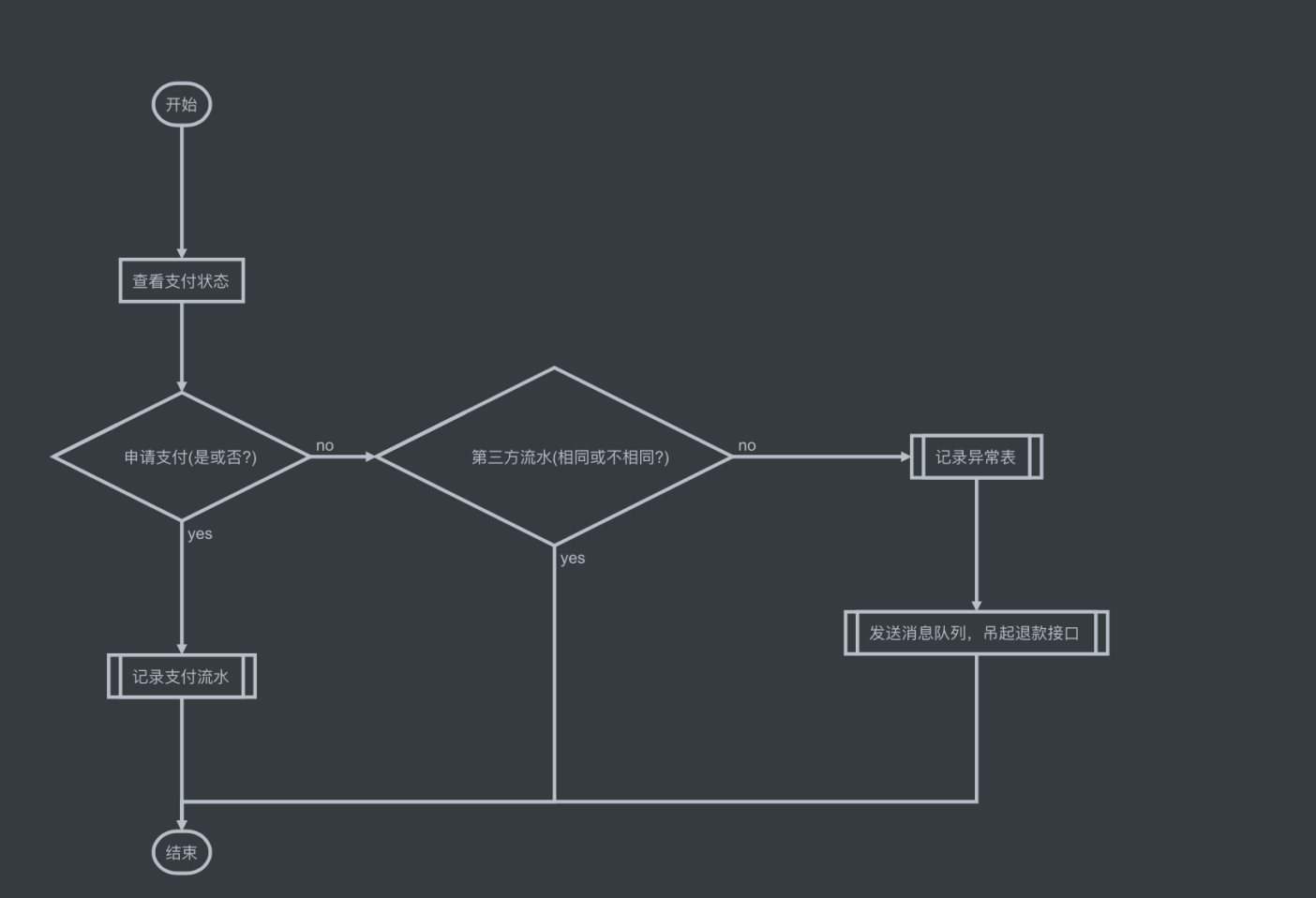


##### 解决方案

在第三方支付中心异步回调时，实现以下功能：

查询支付中心流水表，按订单号查询支付状态是否为申请支付状态，如果是，将此订单支付状态修改为支付完成，记录到支付中心流水表；如果不是申请支付状态，对比支付中心流水表中现有记录的第三方平台交易流水号和第三方平台返回的第三方平台交易流水号，如果两者都相同则不进行任何操作，表示是第三方支付的重复通知，直接返回结果即可；如果两者不同，则记录到异常支付单表中。需要运营人员进行线下退款处理或者直接调用退款接口退款。

流程图如下：



## 8. 秒杀微服务

使用下面的流程图对比代码，多去看几遍，熟悉了就知道秒杀的解决方案了，需要理解。

