### MQ部分必看：

1.体验一下面试官对于消息队列的7个连环炮

2. MQ的作用

3. MQ保证消息队列的高可用

4. MQ保证消息不被重复消费

5. MQ如何保证消息的顺序性

6. MQ如何处理消息丢失的问题

7. MQ消息持续积压几小时怎么处理

### Redis部分必看:

1. Redis线程模型

采用的是多路复用技术，其中epollo的方式进行交互，主要采用的是io 多路复用技术，去监听多个socke文件描述符号fd，将这些fd产生的事件存储到队列中，文件分发器每次都从队列中拿一个数据，把事件交给对方处理器进行处理；这个处理过程是一个线程处理的；

1. Redis数据类型以及用的场景

**String:** 简单的sds结构，底层是通过一个redisObject对象进行存储的，key-value都是String, 当存储的数据是int ，encoding会转换为int;可以用于短信验证嘛，图片验证码

**Hash:** 他就是String类型 filed + value的一个映射表结构。可以存储两种结构，一种是 key+filedname +value的结构，这种简化了序列化和并发的问题，但是会导致浪费内存；

Key+ map的结构；

**List:** 是一个有序集合，可以头插和尾插，lpush,lpop,例如有关注的顺序列表呀，时间相册呀

**Set:** 是一个不能重复的无序集合，内部其实是一个key-value的map结构，value是null值，实际上就是通过hashcode码经行的快速排查重复数据，可以判断一个数据是否存在集合内，而list是不具备的；**提供一些交集并集和差集；**

**场景：**可以用于共同好友，共同拥有的项目呀等

**Sort Set:** 是一个有序的不重复的集合，底层hash，带有一个souce分数进行排序，添加删除，时间复杂度O（1），最大长度为2^32 -1 ,40多亿个；

底层实现：skipList和dict(hash)组成的，hashmap存储key和score (分数）保证数据的顺序，skipList存储的是数据的成员变量，用于快速查询数据；

场景：

1. Redis过期策略

定时过期：

惰性过期：

定期过期三种模式

内存淘汰策略：

Allkey:所有的key

Volatile:设置过期时间的

Random:随机

LRU:最近最少使用

TTL:优先删除剩余时间短的

* noeviction(默认策略)：对于写请求不再提供服务，直接返回错误（DEL请求和部分特殊请求除外）
* allkeys-lru：从所有key中使用LRU算法进行淘汰
* volatile-lru：从设置了过期时间的key中使用LRU算法进行淘汰
* allkeys-random：从所有key中随机淘汰数据
* volatile-random：从设置了过期时间的key中随机淘汰
* volatile-ttl：在设置了过期时间的key中，根据key的过期时间进行淘汰，越早过期的越优先被淘汰
* 当使用volatile-lru、volatile-random、volatile-ttl这三种策略时，如果没有key可以被淘汰，则和noeviction一样返回错误。

1. Redis持久化RDB/AOF

RDB文件：主要是通过快照的方式进行存储,当前线程在某一时刻对对象经行数据存储，通过主线程发送指令，主线程创建调用fork（）生成一个子线程；并且将数据经行全量复制一个快照文件RDB;返回流程（异步操作），主要可以通过bgsave的命令，前提需要配置一些相关配置，例如开启rdb，rdb文件的存储位置，save的方法是同步过程，会阻塞线程；

AOF文件：是一种追加式的文件，存储的式操作命令，重启时会通过执行AOF文件中的命令，将数据重写到内存中，以达到恢复数据的效果；主要的方式有三种：alway:实时存储数据,everying:每秒钟缓存数据到AOF中,no：通过系统控制什么时候经行数据存储，不可控制；这种会消耗性能，如果不叫平凡的话，但是如果要保证数据的完整性能话，只能通过完全是由数据库或是，完全使用redis的方式，如果没有就直接返回错误；怎样保证数据的正确性，只能时通过业务的要求合理的安排；

原理：通过主线程发送的bgwriteAOF指令，fork出一个子线程，这里会发生一种情况就是，当我们同步数据的时候，外部还是会有请求发送过来，aof会单独创建一个临时文件，同步数据还是以前的文件，但是新的命令文件会将数据存储到零时文件中，当数据通过完成后，会将两个aof文件进行合并；替换，aof缓存数据再次写入到文件中，然后替换掉之前的文件，以达到同步的效果；

AO

5. Redis集群模式架构思想/gossip通信协议/hash slot算法/部署经验

6. Redis高可用架构

7. 高并发场景下的缓存一致性问题

8、为什么redis单线程还这么快

1、内存操作，

2、多线程接受处理socket，并不影响性能；redis线程处理 10万+/s的处理速度

3、IO多路复用，减少了IO操作，其实就是多路复用调用epollo和select方法，通过用户态-》内核态访问数据的方式发生了变化，主要还是通过内核的数据copy，减少了Io次数；

### 分库分表必看：

1. 体验一下面试官对于分库分表这个事儿的一个连环炮

2. 垂直拆分/水平拆分

3. 动态扩容/不停机迁移

4. 分库分表后的全局ID方案

熔断技术必看：

1. 资源隔离/熔断/降级的作用

2. 线程池/信号量两种隔离级别技术理解

3. 熔断技术的执行步骤和流程

复习面试突击2内容，包括下面的重点:

### Springcloud部分必看：

1. Springcloud底层架构原理

2. Springcloud与Dubbo的对比

3. Springcloud的组件

### 注册中心部分必看：

1. Eureka注册中心基础原理

**2. Zookeeper注册中心基础原理**

3. Euerka注册中心参数调优

### 分布式事务部分必看：

1. 怎么设计分布式事务技术方案

2. TCC事务、最终一致性事务的技术选型

3. RocketMQ对分布式事务支持的底层实现原理

## 2、复习面试突击3内容，包括下面的重点:

### 2.1、并发编程部分必看：

1. HashMap底层结构/JDK1.8 HashMap的优化/hash算法/寻址算法

2. HashMap如何解决hash碰撞/如何扩容

3. Volatile/synchronized底层原理(原子性、可见性、顺序性)

4. CAS底层原理/CAS会引发的问题

5. AQS底层原理/AQS核心的执行流程

6. 线程池的底层工作原理

7. 线程池的核心配置参数

8. 8.ConcurrentHashMap底层原理

### 2.2、网络部分必看：

1. TCP/IP四层网络模型

2. 浏览器请求的全过程是怎么样的

3. HTTPS的工作原理

### 2.3、Zookeeper部分必看：

1. ZooKeeper可以做什么？

2. ZooKeeper集群的三种角色：Leader、Follower、Observer

3. ZooKeeper最核心的一个机制：Watcher监听回调

4. ZAB的核心思想介绍：主从同步机制和崩溃恢复机制

5. ZooKeeper到底是强一致性还是最终一致性

### 2.4、JVM专栏课内容，包括下面的重点:

1. 类加载机制/类加载过程

类加载过程，组要是通过java的编译器将。Java文件编译成了一个.class的机器编码的文件，加载:

-验证:

准备（主要是为类中的静态变量或是八种基本数据类型赋初值之例如int = 0）

初始化：

使用：

销毁：

2. 双亲委派机制/以及他的作用

3. JVM内存区域(堆、栈等其他)

4. 堆内存划分

5. 新生代、老年代垃圾回收机制以及原理

6. 常用的垃圾回收器(ParNew、CMS、G1)

7. 触发垃圾回收的时机和条件

 Minor GC ，Full GC 触发条件

Minor GC触发条件：当Eden区满时，触发Minor GC。

Full GC触发条件：

（1）调用System.gc时，系统建议执行Full GC，但是不必然执行

（2）**老年代空间不足**

（3）方法去空间不足

（4）通过Minor GC后进入老年代的平均大小大于老年代的可用内存

（5）由Eden区、From Space区向To Space区复制时，对象大小大于To Space可用内存，则把该对象转存到老年代，且老年代的可用内存小于该对象大小

1. JVM调优参数模板

Jvm调优参数:

Xxs:

Xms:

X

9. JVM分析命令(jstat、jmap、jstack等)

10. JVM分析工具(MAT等)

### JVM总结的图：之后汇总PDF给出

Rocketmq专栏课内容，包括下面的重点:

1. Rocketmq架构原理

2. Rocketmq场景优化

3. Rocketmq底层原理/消息持久化/raft协议

4. Rocketmq事务机制

5. Rocketmq消息0丢失

6. Rocketmq重复消息/死信队列

### Mysql专栏课内容，包括下面的重点:

#### 1. Mysql架构设计(SQL接口、查询解析器、查询优化器、存储引擎、执行器)

#### 2. Mysql日志相关(undo log、redo log、binlog)

#### 3. Innodb存储引擎(buffer pool结构、free链表作用、flush链表作用、lru链表作用)

#### 4. Mysql数据页/数据行/表空间/数据区

#### 5. Mysql事务隔离级别/MVCC机制原理

#### 6. 索引原理/索引规则/索引优化

#### 7. Explain执行计划分析

#### 8. 线上问题排查

8.1、线上数据库莫名其妙的随机性能抖动优化?

问题分析，可能的原因是什么

### MYSQL脑图：之后汇总PDF给出

架构课内容：

大型电商系统v1.0项目实战里面的设计模式部分必看：

复习几个比较重要的设计模式：

策略模式/迭代器模式/工厂模式/代理模式/构造器模式/适配器模式

### 基于微服务架构的大型分布式电商系统部分必看：

主要复习微服务Spring Cloud那块：

1. Eureka服务注册与发现的原理/心跳检测/注册表抓取/自我保护机制

2. Eureka注册表多级缓存机制/参数调优

3. Ribbon负载均衡工作原理/负载均衡算法IRule

4. Hystrix资源隔离、限流、熔断、降级

5. Hystrix的线程池隔离技术/信号量隔离技术

6. Hystrix执行时的8大流程步骤以及内部原理

### 分布式事务实践模块部分必看：

1. 事务的基础知识：ACID以及几种隔离级别

2. 事务的基础知识：Spring的事务支持以及传播特性

3. XA规范以及2PC分布式事务理论介绍

4. 2PC分布式事务方案的缺陷以及问题/3PC分布式事务方案的理论知识讲解

5. 理解CAP与BASE的基础知识

6. 理解TCC分布式事务技术方案以及原理/TCC分布式事务执行流程

7. 可靠消息最终一致性方案/整体架构以及核心流程设计

### 分布式系统架构实战部分必看：

1. 保证分布式系统的接口幂等性的几种常见方案

2. Redisson分布式锁原理

3. Redisson可重入锁/lua脚本加锁逻辑/watchdog维持加锁/锁的互斥阻塞/释放锁

4. Redisson公平锁原理

5. MultiLock原理/RedLock源码算法实现

6. Zookeeper Curator框架介绍以及分布式锁支持

7. Zookeeper分布式锁可重入锁源码剖析

8. Zookeeper分布式锁可重入读写锁源码剖析

### JDK源码剖析系列部分必看：

集合部分：

1. ArrayList基本原理以及优缺点

2. ArrayList数组扩容以及元素拷贝

3. LinkedList基本原理以及优缺点

4. LinkedList双向链表数据结构

5. HashMap数组、链表、红黑树的数据结构

6. HashMap优化后的降低冲突概率的hash算法

7. HashMap put操作原理以及hash寻址算法

8. HashMap JDK 1.8引入红黑树优化hash冲突

9. HashMap JDK 1.8的高性能rehash算法

10. LinkedHashMap底层原理/有顺序的map数据结构

11. TreeMap/HashSet/LinkedHashSet/TreeSet底层结构以及原理

12. Iterator迭代器应对多线程并发修改的fail fast机制

### 并发编程部分：

1. CPU多级缓存模型

2. 总线加锁机制和MESI缓存一致性协议

3. Java内存模型

4. volatile是如何保证可见性/顺序性

5. volatile的底层实现原理：lock指令以及内存屏障

6. synchronized底层原理（jvm指令以及monitor锁）

7. 可见性涉及的底层硬件概念：寄存器、高速缓存、写缓冲器

8. 深入探秘有序性：Java程序运行过程中发生指令重排的几个地方

9. synchronized锁同时对原子性、可见性以及有序性的保证

10. 采用写缓冲器和无效队列优化MESI协议的实现性能

11. AtomicInteger中的CAS无锁化原理

12. AtomicInteger源码剖析：仅限JDK内部使用的Unsafe类

13. AtomicInteger源码剖析：底层CPU指令是如何实现CAS语义的

14. Atomic原子类体系的CAS语义存在的三大缺点分析/CAS的引发问题的解决方案

15. AQS的原理(异步队列同步器)/以及AQS重要的参数

16. AQS默认的非公平加锁策略的运作原理

17. AQS队列唤醒阻塞线程的过程

18. ReentractLock如何设置公平锁策略

19. ThreadLocal源码剖析：线程本地副本的实现原理

20. JDK 1.7 HashMap并发环境下死循环之环形链表

21. JDK 1.7 HashMap并发环境下死循环之死循环与丢数据

22. ConcurrentHashMap底层原理：初始化流程

23. ConcurrentHashMap底层原理：未分段数组的CAS加锁

24. ConcurrentHashMap底层原理：链表和红黑树解决hash冲突问题

25. JDK 1.8对ConcurrentHashMap做出的锁细粒度优化

26. CopyOnWriteArrayList：线程安全的List数据结构/基于写时复制机制

27. CopyOnWriteArrayList核心思想：弱一致性提升读并发

28. 线程安全的有界队列：LinkedBlockingQueue

29. 基于数组实现的有界队列：ArrayBlockingQueue

30. 线程池的核心成员变量有哪些/以及作用

31. 线程池的源码执行流程

32. 线程池的几个种类/区别/以及使用场景

### 大白话网络课程部分必看：

1. 数据链路层：以太网协议、mac地址、网卡以及路由器

2. 网络层：IP协议、子网划分以及子网掩码

3. 传输层：TCP协议、Socket编程是什么

4. 应用层：HTTP协议是什么

5. HTTPS协议加密通信的实现原理

6. 全世界几万台DNS服务器如何大接力完成IP地址的查询

7. ARP广播获取路由器mac地址以及ARP缓存机制

河小马(儒猿技术窝管理员):

## 公司的项目准备：

1. 梳理自己做过比较复杂的或者最有挑战性的功能或者项目

2. 把上面的功能或者项目画一下核心的流程图

3. 把上面的功能或者项目一些功能点列一下，还有表结构串联起来

4. 做完这些功能和项目难点在哪，从里面学到什么东西，自己回顾一下

5. 梳理项目的时候，参考闲人大佬之前在群里分享的内容，比如架构班学员如何梳理项目，按照他们的思路去梳理项目。

以上内容基本都是我面试前的准备，基本是我挑选过的一些重点知识。 有些课程之前看过，复习的时候过过笔记就行了，但大多时候，我会有重新看视频或者重新看专栏课的文章。

### 4. 简历准备

当你复习完知识后，你心里面就有数了，哪些知识点比较扎实，比较熟悉。项目里用过哪些技术，线上问题怎么排查等等，这些都是很好的简历素材。

举几个例子：

JVM专栏课里面的知识你都理解透彻了，那么你的简历就可以写：

熟悉JVM底层工作原理和垃圾回收机制，熟悉使用jstat、jmap、mat进行JVM调优

MYSQL专栏课里面的知识你都理解透彻了，那么的简历就可以写：

Mysql熟悉Innodb引擎、MVCC机制，能根据explain执行计划优化SQL

你有读过Redission、zk分布式锁、SpringCloud的源码，那么你的简历就可以写：

对开源技术有一定研究，对Spring Cloud、Redisson 、zk分布式锁等开源框架深入阅读过源码

上面的简历填充就是很好的技巧了，可以丰富你简历里面的个人总结和技术专业技能这些栏目了。

项目经验的话，那么你就要突出项目的技术亮点，技术难点，技术栈，你负责的模块的细节，这些都要根据你实际的项目开发经验来写了，毕竟每个人的项目经验都不一样。

很重要的一点是，我简历的内容，如专业技能和项目经历这些都是找老师帮我定制优化的，他给我提出一些专业技能和项目经历的优化建议，还有怎么丰富简历的内容，反复1-2次的调整与修改，我的简历就会“焕然一新”了。

### 5. 图片准备

1. 画出最近项目的系统架构图，先用processon画一次，再手撸画多几次。

2. 画出项目里面比较复杂的功能或者比较难的功能的流程图，先用processon画一次，再手撸画多几次。

3. 重要的技术流程图也手画了，例如Eureka注册表的多级缓存、AQS核心流程图、可靠消息最终一致性方案流程图、Redis线程模型流程图等等。

准备这些图有什么用呢？手撸给面试官看的。

### 6. 面试内容

根据自己的面试印象回忆，先归纳一下经常问的一些热点问题：

1. 线上某个程序CPU 100%，一般是什么原因导致的？怎么排查？讲讲排查步骤？

### 问题分析：

主要考你JVM的基础知识和一些JVM故障的排查命令。

这些问题很简单，你看完JVM专栏课就知道对应的排查命令了，例如ps java进程的pid、Top -Hp、利用Jstack分析。这种CPU 100%出现的原因有很多，例如代码哪里没写好出现了死循环、一些连接池再调用之后，没有合理关闭等等。

但你只是回答出这些内容，我觉得面试官觉得你只能拿到70分，刚好达到及格线。如果你想拿到90分+的话，必须继续分析出现这种问题的根本原因是啥？

那你就可以继续带出JVM的堆内存相关知识点，例如新生代、老年代有没有合理的分配，java启动参数有没有合理指定大小，垃圾回收器有没有合理指定，出现问题的时候要合理的输出日志，线上可以用过jstat、jmap等命令定位出新生代、老年代的回收次数和哪些对象占用比较大。

一步一步这样分析的话，让面试官知道你已经非常掌握到这块的知识了。

2. 有些sql执行很慢，怎么分析？要怎么优化？

问题分析：

主要考你sql的分析能力和优化能力。

首先要打开mysql的慢查询日志，慢查询日志会统计出系统执行sql后出现的一些慢sql汇总，然后就拿出慢sql来分析了，看看对应的sql查询条件有没有走索引，如果没走索引就新建对应的索引就可以了。

但你只是回答出这些内容，我觉得面试官觉得你只能拿到65分，刚好达到及格线。如果你想拿到90分+的话，必须告诉面试官后面怎么避免出现这种慢sql的问题，还有怎么深度调优？

其实你看了Mysql专栏，都已经知道答案了。

首先可以从innodb存储引擎里面的buffer pool入手，调优对应的buffer pool数量参数，然后继续把索引的结构B+树和原理讲解一次，避免查询的sql频繁的进行回表操作，还有主键的值是否是有序的，不建议是用uuid，如果使用uuid作为主键的后果是啥。然后继续讲一下使用索引的规则，几种索引的类型，怎样命中索引，怎么避免频繁回表。Explain分析sql语句的时候，type命中了哪个类型是最快的，type类型分了哪几类等等，如果你能继续带上分库分表的场景，那就锦上添花了。

一步一步这样分析的话，让面试官觉得你对mysql很熟悉了。

3. 分布式锁你用过哪些？使用的时候要注意什么地方？

问题分析：

主要考察你对redis、redission、zk锁这块的知识点

可以先从redis原生的分布式锁讲起，讲一下他的优缺点

然后再讲redission实现分布式锁的原理和机制：

（1）加锁：在redis里设置hash数据结构，生存周期是30000毫秒

（2）维持加锁：代码里一直加锁，redis里的key会一直保持存活，后台每隔10秒的定时任务（watchdog）不断的检查，只要客户端还在加锁，就刷新key的生存周期为30000毫秒

（3）可重入锁：同一个线程可以多次加锁，就是在hash数据结构中将加锁次数累加1

（4）锁互斥：不同客户端，或者不同线程，尝试加锁陷入死循环等待

然后再讲一下zk锁，zk与redis做分布式锁的区别和优缺点

zk锁的znode结构和核心的watch监听器机制

这里顺便把zk锁的如何防止羊群效应的思想也讲一下

zk的zab协议的大概方案也讲一下

已经回答的很不错了

4. 如何保证幂等性？

这个问题rocketmq专栏课和架构课已经讲的很详细了，这里就不说了

5. 讲下synchronized的锁升级

这个问题我答不出来，主要是synchronized的锁升级没复习到这块，其实架构课里面都有讲，然后跟面试官卖惨，说没啥印象了，因为前面的synchronized原理已经回答的差不多了。

面试后，我就马上的回顾了这块的知识了，面试不懂的问题，自己要记录下来，然后复习一下。

（1）锁消除

锁消除是JIT编译器对synchronized锁做的优化，在编译的时候，JIT会通过逃逸分析技术，来分析synchronized锁对象，是不是只可能被一个线程来加锁，没有其他的线程来竞争加锁，这个时候编译就不用加入monitorenter和monitorexit的指令。

（2）锁粗化

（3）偏向锁

（4）轻量级锁

（5）适应性锁

6. 讲下可靠消息最终一致性方案

这个基本是手撸画图给面试官看的，边画边讲解，详细的内容可以看看rocketmq专栏课和分布式事务的架构课内容

7. Eureka注册中心和Nacos注册中心和Zookeeper注册中心的区别

讲一下Eureka注册中心是如何保证AP的/Eureka注册表的多级缓存机制。

讲一下Zookeeper是如何保证CP的。

讲一下nacos是怎样支持AP/CP的，顺便讲一下nacos集成了配置中心的功能。

这里说一下，我对nacos不是很熟悉，但知道他的大概特性，讲nacos的时候顺便也带一下现在比较流行的Spring Cloud Alibaba那套，面试前也复习了一下nacos的思想，应付面试官毫无压力。

根据自己的面试印象回忆，讲一下一些开放性题目：

#### 1. 讲一下你自己做的最复杂或者最难的功能。

这个要根据自己的项目经验和业务模块来讲，我的建议最好是挑最近的公司的项目来准备，因为最近公司的项目根据目前时间来说你是最熟悉的。

你自己要准备项目里面的1-2个比较复杂的功能，要理解里面的细节，最好自己手画一下流程图，因为问这种问题的面试官，往往会抠你项目的细节，如果你不知道项目的细节的话，你会被面试官问的不知道如何回答，这时候就会很尴尬。

#### 2. 如果要你自己设计一个rpc框架，你要考虑哪些点？

这个主要考察你对框架设计的总体能力，例如框架的可用性，可扩展性，可读性，可伸缩性

然后也讲讲市面上一些比较流量的rpc框架，不同rpc框架他们之间的区别，以及他们是怎么保持上面的几个性的

#### **3. 如何保证系统的高可用、高并发？**

这种问题就最好回答了，看过专栏课、架构课后，这些问题其实没啥压力。

可以从最上层开始慢慢的剖析，如服务器用了Lvs + nginx + keepalive的负载均衡架构，LVS服务器，作为负载均衡调度器，请求到来了以后，直接基于linux内核内部的一些底层机制进行请求转发，转发给后端的Web服务器池子里的一台服务器，本质上就是在进行负载均衡，而且因为LVS主要是基于linxu内核级的一些底层机制做请求转发，所以说效率极高，单机抗每秒几十万甚至上百万吞吐量绝对不是问题。

然后就带出MQ异步、MQ削峰、防止消息堆积，幂等性处理、多级缓存、Redis Cluster、防止缓存雪崩、防止缓存穿透、防止缓存击穿、熔断降级技术、分布式锁、分布式调度、分库分表等。