**一、开场白**

简单的介绍一下自己的工作经历与职责，在校或者工作中主要的工作内容，主要负责的内容；（你的信息一清二白的写在简历上，这个主要为了缓解面试者的压力）

介绍下自己最满意的，有技术亮点的项目或平台，重点介绍下自己负责那部分的技术细节；（主要考察应聘者对自己做过的事情是否有清晰的描述，判断做的事情的复杂度）

**二、Java多线程**

**1.线程池的原理，为什么要创建线程池？创建线程池的方式?**

线程池的几个主要参数的作用

public ThreadPoolExecutor(int corePoolSize,

int maximumPoolSize,

long keepAliveTime,

TimeUnit unit,

BlockingQueue<Runnable> workQueue,

ThreadFactory threadFactory,

RejectedExecutionHandler handler)

corePoolSize: 规定线程池有几个线程(worker)在运行。

maximumPoolSize: 当workQueue满了,不能添加任务的时候，这个参数才会生效。规定线程池最多只能有多少个线程(worker)在执行。

keepAliveTime: 超出corePoolSize大小的那些线程的生存时间,这些线程如果长时间没有执行任务并且超过了keepAliveTime设定的时间，就会消亡。

unit: 生存时间对于的单位

workQueue: 存放任务的队列

threadFactory: 创建线程的工厂

handler: 当workQueue已经满了，并且线程池线程数已经达到maximumPoolSize，将执行拒绝策略。

java.util.concurrent.Executors提供了一个 java.util.concurrent.Executor接口的实现用于创建线程池,

①newSingleThreadExecutor

单个线程的线程池，即线程池中每次只有一个线程工作，单线程串行执行任务

②newFixedThreadExecutor(n)

固定数量的线程池，没提交一个任务就是一个线程，直到达到线程池的最大数量，然后后面进入等待队列直到前面的任务完成才继续执行

③newCacheThreadExecutor（推荐使用）

可缓存线程池，当线程池大小超过了处理任务所需的线程，那么就会回收部分空闲（一般是60秒无执行）的线程，当有任务来时，又智能的添加新线程来执行。

④newScheduleThreadExecutor

大小无限制的线程池，支持定时和周期性的执行线程

1. **线程的生命周期，什么时候会出现僵死进程；**

线程有五个状态，分别是新建(New)、就绪(Runnable)、运行(Running)、阻塞(Blocked)和死亡(Dead)。

GC时间过长或者文件句柄数占用过高

1. **说说线程安全问题，什么实现线程安全，如何实现线程安全；**

多个线程同时操作同一个资源时，如果没有加锁，可能出现线程安全问题，最长见就是数字增加，

同时满足以下两个条件时：

多个线程在操作共享的数据。  
操作共享数据的线程代码有多条。

就是线程同步的意思，就是当一个程序对一个线程安全的方法或者语句进行访问的时候，其他的不能再对他进行操作了，必须等到这次访问结束以后才能对这个线程安全的方法进行访问  
什么叫线程安全：  
如果你的代码所在的进程中有多个线程在同时运行，而这些线程可能会同时运行这段代码。如果每次运行结果和单线程运行的结果是一样的，而且其他的变量的值也和预期的是一样的，  
就是线程安全的。  
或者说:一个类或者程序所提供的接口对于线程来说是原子操作或者多个线程之间的切换不会导致该接口的执行结果存在二义性,也就是说我们不用考虑同步的问题。  
线程安全问题都是由全局变量及静态变量引起的。  
若每个线程中对全局变量、静态变量只有读操作，而无写操作，一般来说，这个全局变量是线程安全的；若有多个线程同时执行写操作，一般都需要考虑线程同步，否则就可能影响线程安全。

1. **创建线程池有哪几个核心参数？ 如何合理配置线程池的大小？**

public ThreadPoolExecutor(int corePoolSize,

int maximumPoolSize,

long keepAliveTime,

TimeUnit unit,

BlockingQueue<Runnable> workQueue,

ThreadFactory threadFactory,

RejectedExecutionHandler handler)

corePoolSize: 规定线程池有几个线程(worker)在运行。

maximumPoolSize: 当workQueue满了,不能添加任务的时候，这个参数才会生效。规定线程池最多只能有多少个线程(worker)在执行。

keepAliveTime: 超出corePoolSize大小的那些线程的生存时间,这些线程如果长时间没有执行任务并且超过了keepAliveTime设定的时间，就会消亡。

unit: 生存时间对于的单位

workQueue: 存放任务的队列

threadFactory: 创建线程的工厂

1. **volatile、ThreadLocal的使用场景和原理；**

ThreadLocal提供了线程独有的局部变量，可以在整个线程存活的过程中随时取用，极大地方便了一些逻辑的实现。常见的ThreadLocal用法有：

- 存储单个线程上下文信息。比如存储id等；

- 使变量线程安全。变量既然成为了每个线程内部的局部变量，自然就不会存在并发问题了；

- 减少参数传递。比如做一个trace工具，能够输出工程从开始到结束的整个一次处理过程中所有的信息，从而方便debug。由于需要在工程各处随时取用，可放入ThreadLocal。

ThreadLocal是各线程将值存入该线程的map中，以ThreadLocal自身作为key，需要用时获得的是该线程之前存入的值

个人认为使用ThreadLocal的场景最好满足两个条件，一是该对象不需要在多线程之间共享；二是该对象需要在线程内被传递。

1. **ThreadLocal什么时候会出现OOM的情况？为什么？**
2. **synchronized、volatile区别、synchronized锁粒度、模拟死锁场景、原子性与可见性；**
3. **JVM相关JVM内存模型，GC机制和原理；**
4. **GC分哪两种，Minor GC 和Full GC有什么区别？什么时候会触发Full GC？分别采用什么算法？**
5. **JVM里的有几种classloader，为什么会有多种？**
6. **什么是双亲委派机制？介绍一些运作过程，双亲委派模型的好处；**
7. **什么情况下我们需要破坏双亲委派模型；**
8. **常见的JVM调优方法有哪些？可以具体到调整哪个参数，调成什么值？**
9. **JVM虚拟机内存划分、类加载器、垃圾收集算法、垃圾收集器、class文件结构是如何解析的；**
10. **Java IO,NIO,以及select poll epoll**

https://blog.csdn.net/u013096088/article/details/79122671

https://blog.csdn.net/davidsguo008/article/details/73556811

**四、Java扩展篇**

1. **红黑树的实现原理和应用场景；**
2. **NIO是什么？适用于何种场景？**
3. **Java9比Java8改进了什么；**
4. **HashMap内部的数据结构是什么？底层是怎么实现的？（还可能会延伸考察ConcurrentHashMap与HashMap、HashTable等，考察对技术细节的深入了解程度）；**
5. **说说反射的用途及实现，反射是不是很慢，我们在项目中是否要避免使用反射；**

由于是本地方法调用，让JVM无法优化(还有JIT？)。反射方法调用还有验证过程和参数问题，参数需要装箱拆箱、需要组装成Object[]形式、异常的包装等等问题，篇幅问题这里不加以叙述。

Class.forName，调用本地方法，耗时

Class.getMethod，遍历该类的共有方法，匹配不到，遍历父类共有方法， 耗时，getMethod会返回得到结果的拷贝，应避免getMethods和getDeclardMethods方法，减少不必要堆空间消耗。

Method.invoke

不要过于频繁地使用反射，大量地使用反射会带来性能问题；

通过反射直接访问实例会比访问方法快很多，所以应该优先采用访问实例的方式。

1. **说说自定义注解的场景及实现**
2. **List 和 Map 区别，Arraylist 与 LinkedList 区别，ArrayList 与 Vector 区别；**

**五、Spring相关**

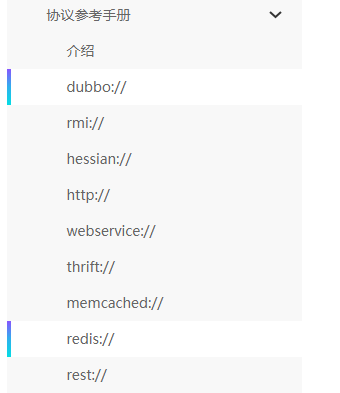
1. **Spring AOP的实现原理和场景？**
2. **Spring bean的作用域和生命周期；**
3. **Spring Boot比Spring做了哪些改进？ Spring 5比Spring4做了哪些改进；**
4. **如何自定义一个Spring Boot Starter？**
5. **Spring IOC是什么？优点是什么？**
6. **SpringMVC、动态代理、反射、AOP原理、事务隔离级别；**

**六、中间件篇**

1. **Dubbo完整的一次调用链路介绍；**
2. **Dubbo支持几种负载均衡策略？**

缺省是随机 random。还可以配置轮询 roundrobin、最不活跃优先 [[4]](http://dubbo.apache.org/zh-cn/docs/user/recommend.html" \l "fn4) leastactive 和一致性哈希 consistenthash 等

1. **Dubbo 支持的协议**



1. **Dubbo Provider服务提供者要控制执行并发请求上限，具体怎么做？**



1. **Dubbo启动的时候支持几种配置方式？**

根据驱动方式的不同（比如Spring或裸API编程）配置形式上肯定会有所差异，具体请参考[XML配置](http://dubbo.apache.org/zh-cn/docs/user/configuration/xml.html)、[Annotation配置](http://dubbo.apache.org/zh-cn/docs/user/configuration/annotation.html)、[API配置](http://dubbo.apache.org/zh-cn/docs/user/configuration/api.html)、属性配置(dubbo.properties)三篇文档。

1. **了解几种消息中间件产品？各产品的优缺点介绍；**
2. **消息中间件如何保证消息的一致性和如何进行消息的重试机制？**
3. **Zookeeper选举机制(https://www.cnblogs.com/lanqiu5ge/p/9405601.html)**
4. **Kafka为什么如此高性能(https://www.jianshu.com/p/ff7dd5b349f1)**
5. Kafka的用途有哪些？使用场景如何？
6. Kafka中的ISR、AR又代表什么？ISR的伸缩又指什么
7. Kafka中的HW、LEO、LSO、LW等分别代表什么？
8. Kafka中是怎么体现消息顺序性的？
9. Kafka中的分区器、序列化器、拦截器是否了解？它们之间的处理顺序是什么？
10. Kafka生产者客户端的整体结构是什么样子的？
11. Kafka生产者客户端中使用了几个线程来处理？分别是什么？
12. Kafka的旧版Scala的消费者客户端的设计有什么缺陷？
13. “消费组中的消费者个数如果超过topic的分区，那么就会有消费者消费不到数据”这句话是否正确？如果不正确，那么有没有什么hack的手段？
14. 消费者提交消费位移时提交的是当前消费到的最新消息的offset还是offset+1?
15. 有哪些情形会造成重复消费？
16. 那些情景下会造成消息漏消费？
17. KafkaConsumer是非线程安全的，那么怎么样实现多线程消费？
18. 简述消费者与消费组之间的关系
19. 当你使用kafka-topics.sh创建（删除）了一个topic之后，Kafka背后会执行什么逻辑？
20. topic的分区数可不可以增加？如果可以怎么增加？如果不可以，那又是为什么？
21. topic的分区数可不可以减少？如果可以怎么减少？如果不可以，那又是为什么？
22. 创建topic时如何选择合适的分区数？
23. Kafka目前有那些内部topic，它们都有什么特征？各自的作用又是什么？
24. 优先副本是什么？它有什么特殊的作用？
25. Kafka有哪几处地方有分区分配的概念？简述大致的过程及原理
26. 简述Kafka的日志目录结构
27. Kafka中有那些索引文件？
28. 如果我指定了一个offset，Kafka怎么查找到对应的消息？
29. 如果我指定了一个timestamp，Kafka怎么查找到对应的消息？
30. 聊一聊你对Kafka的Log Retention的理解
31. 聊一聊你对Kafka的Log Compaction的理解
32. 聊一聊你对Kafka底层存储的理解（页缓存、内核层、块层、设备层）
33. 聊一聊Kafka的延时操作的原理
34. 聊一聊Kafka控制器的作用
35. 消费再均衡的原理是什么？（提示：消费者协调器和消费组协调器）
36. Kafka中的幂等是怎么实现的
37. Kafka中的事务是怎么实现的（这题我去面试6加被问4次，照着答案念也要念十几分钟，面试官简直凑不要脸）
38. Kafka中有那些地方需要选举？这些地方的选举策略又有哪些？
39. 失效副本是指什么？有那些应对措施？
40. 多副本下，各个副本中的HW和LEO的演变过程
41. 为什么Kafka不支持读写分离？
42. Kafka在可靠性方面做了哪些改进？（HW, LeaderEpoch）
43. Kafka中怎么实现死信队列和重试队列？
44. Kafka中的延迟队列怎么实现（这题被问的比事务那题还要多！！！听说你会Kafka，那你说说延迟队列怎么实现？）
45. Kafka中怎么做消息审计？
46. Kafka中怎么做消息轨迹？
47. Kafka中有那些配置参数比较有意思？聊一聊你的看法
48. Kafka中有那些命名比较有意思？聊一聊你的看法
49. Kafka有哪些指标需要着重关注？
50. 怎么计算Lag？(注意read\_uncommitted和read\_committed状态下的不同)
51. Kafka的那些设计让它有如此高的性能？
52. Kafka有什么优缺点？
53. 还用过什么同质类的其它产品，与Kafka相比有什么优缺点？
54. 为什么选择Kafka?
55. 在使用Kafka的过程中遇到过什么困难？怎么解决的？
56. 怎么样才能确保Kafka极大程度上的可靠性？
57. 聊一聊你对Kafka生态的理解
58. **Tomcat调优参数，常见问题(https://blog.csdn.net/chengxuyuan\_110/article/details/81033969)**

**七、数据库篇**

1. **锁机制介绍：行锁、表锁、排他锁、共享锁；**
2. **乐观锁的业务场景及实现方式；**
3. **事务介绍，分布式事物的理解，常见的解决方案有哪些，什么事两阶段提交、三阶段提交；**
4. **MySQL记录binlog的方式主要包括三种模式？每种模式的优缺点是什么？**
5. **MySQL锁，悲观锁、乐观锁、排它锁、共享锁、表级锁、行级锁；**
6. **分布式事务的原理2阶段提交，同步\异步\阻塞\非阻塞；**

数据库事务隔离级别，MySQL默认的隔离级别、Spring如何实现事务、JDBC如何实现事务、嵌套事务实现、分布式事务实现；

SQL的整个解析、执行过程原理、SQL行转列；

**八、Redis**

## 1、什么是Redis?

Redis 是完全开源免费的，遵守BSD协议，是一个高性能的key-value数据库。

## Redis 与其他 key - value 缓存产品有以下三个特点：

Redis支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用。 Redis不仅仅支持简单的key-value类型的数据，同时还提供list，set，zset，hash等数据结构的存储。 Redis支持数据的备份，即master-slave模式的数据备份。

## Redis 优势

性能极高 – Redis能读的速度是110000次/s,写的速度是81000次/s 。 丰富的数据类型 – Redis支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。 原子 – Redis的所有操作都是原子性的，意思就是要么成功执行要么失败完全不执行。单个操作是原子性的。多个操作也支持事务，即原子性，通过MULTI和EXEC指令包起来。 丰富的特性 – Redis还支持 publish/subscribe, 通知, key 过期等等特性。

## Redis与其他key-value存储有什么不同？

Redis有着更为复杂的数据结构并且提供对他们的原子性操作，这是一个不同于其他数据库的进化路径。Redis的数据类型都是基于基本数据结构的同时对程序员透明，无需进行额外的抽象。

Redis运行在内存中但是可以持久化到磁盘，所以在对不同数据集进行高速读写时需要权衡内存，因为数据量不能大于硬件内存。在内存数据库方面的另一个优点是，相比在磁盘上相同的复杂的数据结构，在内存中操作起来非常简单，这样Redis可以做很多内部复杂性很强的事情。同时，在磁盘格式方面他们是紧凑的以追加的方式产生的，因为他们并不需要进行随机访问。

## 2、Redis的数据类型？

答：Redis支持五种数据类型：string（字符串），hash（哈希），list（列表），set（集合）及zsetsorted set：有序集合)。

我们实际项目中比较常用的是string，hash如果你是Redis中高级用户，还需要加上下面几种数据结构HyperLogLog、Geo、Pub/Sub。

如果你说还玩过Redis Module，像BloomFilter，RedisSearch，Redis-ML，面试官得眼睛就开始发亮了。

## 3、使用Redis有哪些好处？

1、速度快，因为数据存在内存中，类似于HashMap，HashMap的优势就是查找和操作的时间复杂度都是O1)

2、支持丰富数据类型，支持string，list，set，Zset，hash等

3、支持事务，操作都是原子性，所谓的原子性就是对数据的更改要么全部执行，要么全部不执行

4、丰富的特性：可用于缓存，消息，按key设置过期时间，过期后将会自动删除

## 4、Redis相比Memcached有哪些优势？

1、Memcached所有的值均是简单的字符串，redis作为其替代者，支持更为丰富的数据类  
2、Redis的速度比Memcached快很  
3、Redis可以持久化其数据

## 5、Memcache与Redis的区别都有哪些？

1、存储方式 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。 Redis有部份存在硬盘上，这样能保证数据的持久性。

2、数据支持类型 Memcache对数据类型支持相对简单。 Redis有复杂的数据类型。

3、使用底层模型不同 它们之间底层实现方式 以及与客户端之间通信的应用协议不一样。 Redis直接自己构建了VM 机制 ，因为一般的系统调用系统函数的话，会浪费一定的时间去移动和请求。

## 6、Redis是单进程单线程的？

答：Redis是单进程单线程的，redis利用队列技术将并发访问变为串行访问，消除了传统数据库串行控制的开销。

## 7、一个字符串类型的值能存储最大容量是多少？

答：512M

## 8、Redis的持久化机制是什么？各自的优缺点？

## Redis提供两种持久化机制RDB和AOF机制:

1、RDBRedis DataBase)持久化方式： 是指用数据集快照的方式半持久化模式)记录redis数据库的所有键值对,在某个时间点将数据写入一个临时文件，持久化结束后，用这个临时文件替换上次持久化的文件，达到数据恢复。

## 优点：

1、只有一个文件dump.rdb，方便持久化。

2、容灾性好，一个文件可以保存到安全的磁盘。

3、性能最大化，fork子进程来完成写操作，让主进程继续处理命令，所以是IO最大化。使用单独子进程来进行持久化，主进程不会进行任何IO操作，保证了redis的高性能) 4.相对于数据集大时，比AOF的启动效率更高。

## 缺点：

1、数据安全性低。RDB是间隔一段时间进行持久化，如果持久化之间redis发生故障，会发生数据丢失。所以这种方式更适合数据要求不严谨的时候)

2、AOFAppend-only file)持久化方式： 是指所有的命令行记录以redis命令请求协议的格式完全持久化存储)保存为aof文件。

## 优点：

1、数据安全，aof持久化可以配置appendfsync属性，有always，每进行一次命令操作就记录到aof文件中一次。

2、通过append模式写文件，即使中途服务器宕机，可以通过redis-check-aof工具解决数据一致性问题。

3、AOF机制的rewrite模式。AOF文件没被rewrite之前（文件过大时会对命令进行合并重写），可以删除其中的某些命令（比如误操作的flushall）)

## 缺点：

1、AOF文件比RDB文件大，且恢复速度慢。 2、数据集大的时候，比rdb启动效率低。

## 9、Redis常见性能问题和解决方案：

1、Master最好不要写内存快照，如果Master写内存快照，save命令调度rdbSave函数，会阻塞主线程的工作，当快照比较大时对性能影响是非常大的，会间断性暂停服务 2、如果数据比较重要，某个Slave开启AOF备份数据，策略设置为每秒同步一 3、为了主从复制的速度和连接的稳定性，Master和Slave最好在同一个局域网 4、尽量避免在压力很大的主库上增加从 5、主从复制不要用图状结构，用单向链表结构更为稳定，即：Master <- Slave1 <- Slave2 <- Slave3…这样的结构方便解决单点故障问题，实现Slave对Master的替换。如果Master挂了，可以立刻启用Slave1做Master，其他不变。

## 10、redis过期键的删除策略？

1、定时删除:在设置键的过期时间的同时，创建一个定时器timer). 让定时器在键的过期时间来临时，立即执行对键的删除操作。

2、惰性删除:放任键过期不管，但是每次从键空间中获取键时，都检查取得的键是否过期，如果过期的话，就删除该键;如果没有过期，就返回该键。

3、定期删除:每隔一段时间程序就对数据库进行一次检查，删除里面的过期键。至于要删除多少过期键，以及要检查多少个数据库，则由算法决定。

## 11、Redis的回收策略（淘汰策略）?

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

注意这里的6种机制，volatile和allkeys规定了是对已设置过期时间的数据集淘汰数据还是从全部数据集淘汰数据，后面的lru、ttl以及random是三种不同的淘汰策略，再加上一种no-enviction永不回收的策略。

Redis为什么这么快？redis采用多线程会有哪些问题？

Redis支持哪几种数据结构；

Redis跳跃表的问题；

Redis单进程单线程的Redis如何能够高并发?

Redis如何使用Redis实现分布式锁？

Redis分布式锁操作的原子性，Redis内部是如何实现的？

**九、其他**

看过哪些源代码？然后会根据你说的源码问一些细节的问题？（这里主要考察面试者是否对技术有钻研的精神，还是只停留在表面，还是背了几道面经，这个对于很多有强迫症的面试官，如果你连源码都没看过，基本上是会pass掉的，比如我也是这样的！）

项目中遇到了哪些比较有挑战性的问题，是如何解决的；（这个很有争议，一方面是你连一个复杂的问题都解决不了，要你过来干什么，还有就是我的能力牛逼啊，但是公司没有业务场景让我展示啊！这个就看你遇到的面试官了，祝你成功