高频面试题

面试题重点要看 mysql，java 基础，spring，jvm，juc，redis，以及使用的一些场景相关的。像 springcloud，springboot，mq 这块不要深究，了解基础常见面试题就行了，毕竟没深入研究过问难的也难回答。

该套面试题涉及一些关键字和一些技术的真实使用场景都有涉及，比如 volicatile 关键字（参考之前多线程导入），syschronize 关键字，线程池，反射，threadlocal，redis 的 5 种数据结构真实项目使用，以及后续的一系列衍生出的问题

1. 说一个栈溢出的例子

用方法无限递归

因为栈存放栈帧，所以使用没有退出条件的递归即可

```
//栈溢出示例
public class StackOverflowTest {
    public static void main(String[] args) {
        new StackOverflowTest().test();
    }

    // 递归调用
    public void test() {
        test();
    }
}
```

3. 开发中要遵循哪些开发规范

1.命名规范，例如类名采用大驼峰式命名法，即首字母大写，后续单词首字母大写

2.注释，在关键地方添加注释，包括方法、类和字段的说明

3.异常处理，避免在 try-catch 块中处理多个不相关的异常，应该将它们分开处理。捕获了异常后不要忽略异常，至少应该将异常记录到日志中

4.类设计，类设计要遵循 oop 思想

5 线程池使用，需要使用可配置参数的自定义线程池，还不要通过 Executors 直接获取的线程池
6. 对可能为空的返回结果要进行判空后再进行业务操作，不然可能出现空指针问题

4. 说一下 arraylist，hashmap，字符串操作常用的 api 有哪些

Arraylist size(),add(),addAll(),get(),sublist(),toArray(),remove()
hashMap get(),put(),keySet(),remove(),containtskey(),putIfAbsent()，
字符串 equals(),endswith(),replace(),replaceAll(),tostring,containts(),substring(),split()

5. jdk1.8 新特性有哪些

1. Lambda 表达式：Lambda 表达式是一种轻量级的匿名函数，可以作为一个函数式接口的实现。

2. Stream API：Stream API 提供了一种流式编程的方式来处理集合或数组中的数据。
3. 新的时间日期 API：新的时间日期 API(如 LocalDate、LocalTime 和 Instant)提供了更好的 API 支持和线程安全性。
4. 默认方法：默认方法为接口添加了一种默认实现，使得接口的实现更加灵活。
5. **函数式接口**：函数式接口是只包含一个抽象方法的接口，Lambda 表达式可以实现这种接口。

6. 怎么把一个集合转成一个 map？

1. 可以通过遍历集合然后存入 map 中
可以用 jdk1.8 的 stream 流，ap<String, String> uuidNameMap = persons.stream()
.collect(Collectors.toMap(Person::getUuid, Person::getName));

7. for 和 foreach 有什么区别？

1. 、for 可以通过 break 关键词来终止循环的执行，而 foreach 不可以；2、for 可以通过控制循环变量的数值来控制循环的执行，而 foreach 不行；3.、for 在循环外可以调用循环变量，而 foreach 在循环外不能调用循环变量；4.、for 的执行效率要高于 foreach。

8. redis 查看 key 还有多少时间过期用什么命令？

ttl key 查看 rediskey 还有多久过期

9. redis 配置文件有哪些配置项(记 5 到 6 个左右)

Bind　　绑定的主机地址，只有 bind 指定的 IP，才能用 redis-cli 登录该 redis 服务器并进行操作。

Port　　监听的端口，默认 6379。

Requirepass　设置 Redis 连接/登录密码。默认关闭

Timeout　多久（秒）未操作后关闭连接

Loglevel　日志级别。默认为 notice

Databases　　管理的数据库数量。默认 0。

Maxclients　　同一时间最大客户端连接数，默认为 0，无限制。

Maxmemory　redis 最大内存限制

还可以设置持久化方式和策略

还可以配置集群模式和刷盘机制以及心跳发送频率

10. redis 优化

1. 可以使用 redis 管道功能，对于一次性执行多个命令和 redis 建立多次连接的情况可以将命令放入管道中，这样和 redis 建立一次连接就可以
2. 对于大 key 问题可以进行拆解优化
3. 尽量不使用 O(N) 以上复杂度过高的命令，对于数据的聚合操作，放在客户端做。
4. 集中过期优化，对 key 增加一个随机过期时间，把集中过期的时间打散，降低 Redis 清理过期 key 的压力

5. 淘汰策略尽量改为随机淘汰，随机淘汰比 LRU 要快很多，因为 lru 会有一个取值比较的过程

6. 如果业务对数据完整性要求不高，可以将持久化机制改成主线程每次写操作只写内存就返回，内存数据什么时候刷到磁盘，交由操作系统决定

11. 你们公司用的什么 redis 版本？

我们公司是用的 springboot2.5.3，springboot-start-data-redis 默认的版本

12. 你们公司用的什么 springboot 版本？

2.5.3

13. 你们公司用的什么 mysql 版本？

8.0.23

14. Mysql8.0 有哪些新特性？（记住 3,4 个即可）

1.MySQL 8 中新增了 隐藏索引 和 降序索 引 。隐藏索引可以用来测试去掉索引对查询性能的影响。在查询中混合存在多列索引时，使用降序索引 可以提高查询的性能。
2.MySQL 8 增加 了聚合函数 JSON_ARRAYAGG() 和 JSON_OBJECTAGG() ，将参数聚合为 JSON 数组或对象
3.在 MySQL 8 版本中支持原子数据定义语言（DDL），提高了数据安全性，对事务提供更好的 支持。
4.MySQL 8 中默认的字符集由 latin1 更改为 utf8mb4
5.窗口函数 MySQL 8 开始支持窗口函数。在之前的版本中已存在的大部分 聚合函数 在 MySQL 8 中也可以 作为窗口函数来使用。

15. Mysql sql 语句执行顺序？

1. 书写顺序

select->distinct->from->join->on->where->group by->having->order by->limit

2. 执行顺序

from->on->join->where->group by->sum、count、max、avg->having->select->distinct->order by->limit

2.说几个堆溢出的例子

循环创建对象，或者一次性从数据库查出很多数据

堆中存放对象实例，只要 new 创建的实例足够大就能溢出。这里使用的是 List 集合存放实例化的 byte 数组，如果不用 List 存放直接使用 new 关键字实例化将无法实现溢出。

因为 new 实例化对象没有地方调用会被垃圾回收机制回收，堆就很难或不会溢出了

```
// 堆溢出实例
public class HeadOverTest {
    public static void main(String[] args) {
        int i = 1;          //记录第几次创建时溢出
        List<byte[]> list = new ArrayList<>();
        while (true) {
            System.out.println("执行次数: "+(i++));
            list.add(new byte[1024*1024*1024]);
        }
    }
}
```

解决方法

增加 jvm 的内存大小，使用 -Xmx 和 -Xms 来设置

检查代码中是否有死循环或递归调用。

检查是否有大循环重复产生新对象实体。

检查对数据库查询中，是否有一次获得全部数据的查询。一般来说，如果一次取十万条记录到内存，就可能引起内存溢出。这个问题比较隐蔽，在上线前，数据库中数据较少，不容易出问题，上线后，数据库中数据多了，一次查询就有可能引起内存溢出。因此对于数据库查询尽量采用分页的方式查询。

检查 List、Map 等集合对象是否有使用完后，未清除的问题。List、MAP 等集合对象会始终存有对对象的引用，使得这些对象不能被 GC 回收。

3. 说一个元空间溢出的例子

循环利用反射创建对象

正如我们在上一章中所解释的，元空间的使用与加载到JVM中的类的数量密切相关。以下代码是最直接的示例：

```
public class Metaspace {
  static javassist.ClassPool cp = javassist.ClassPool.getDefault();

  public static void main(String[] args) throws Exception{
    for (int i = 0; ; i++) {
      Class c = cp.makeClass("eu.plumbr.demo.Generated" + i).toClass();
    }
  }
}
```

在本例中，源代码迭代循环并在运行时生成类。所有这些生成的类定义最终都会消耗元空间。类生成的复杂性由javassist库处理。

解决办法是什么？

当面对元空间导致的 OutOfMemoryError 时，更改应用程序启动配置并增加以下内容：

```
-XX:  MaxMetaspaceSize=512m
```

4. 你知道什么是跨域问题吗？

浏览器为了让自己更安全，设置了一种同源策略，"协议+域名+端口"三者都相同才算同源，不然会阻止一些请求。

5. 如何解决跨域问题？

（1）创建一个过滤器，然后把一些来源，请求头和请求方法等都设置成全部允许

（2）可以在类或者方法上面加@CrossOrigin 注解

（3）还可以在 nginx 配置文件配置允许跨域

（4）网关也可以统一配置跨域

4. 你知道哪些加密算法

（1）我知道有 MD5 算法，MD5 算法主要用的是 哈希函数，不可逆，无论传入多长的字符串，MD5 都会输出长度为 128bits 的一个串。

（2）还有 DES 加密算法，他的话是主要是 64 位为分组对数据加密，它的密钥长度 是 56 位，加密解密用同一算法。

（3）对 RSA 也了解一点，这种加密算法是将两个大素数相乘很容易，但想要对这个乘积进行因式分解就比较困难了，所以一般将乘积作为加密密钥。

5. 如何保证你写的接口的安全性

最好使用 https 协议，在 http 和 tcp 之间添加一层加密层(SSL 层)，

这一层负责数据的加密和解密，这样就会防止数据被抓包；

第二种做法可以在每次请求中加入当前的时间，服务器端会拿到当前时间和消息中的时间相减，看看是否在一个固定的时间范围内比如 5 分钟内；这样恶意请求的数据包是无法更改里面时间的，所以 5 分钟后就视为非法请求。

加签名:比如使用 md5 算法，将需要提交的数据通过某种方式组合和一个字符串，然后通过 md5 生成一段加密字符串，客户端和服务端都计算一份，防止数据传输被篡改。

服务器黑白名单：对接口的访问 ip 进行黑白名单进行配置

6. sql 语句什么时候会出现 filesort

1.sql 有 order by 或者 group by 没有符合最左匹配原则。

2.出现了隐式类型转换。

7. Springboot 动态切换数据源这个你知道吗

这个我切换过，需要继承 Spring 的 AbstractRoutingDataSource 抽象类，然后重写 determineCurrentLookupKey()这个方法（可以说是重写一个什么什么 key 方法），因为这个方法是设置数据源的，然后这个方法里面 return 一个数据源，这个数据源我们可以事先配几个数据源，然后程序启动加载全部数据源设置进子类里面，最后根据就可以根据传值获取想要的数据源了。

8. Arraylist 在多线程下会出现什么异常

ConcurrentModificationException（并发修改异常）

为什么：

在多个线程进行 add 操作时可能会导致 elementData 数组越界，还会导致一个线程的值覆盖另一个线程添加的值

那怎么处理

（1）使用 Vector 容器，也是加了 synchronizedList

（2）使用 Collections 的静态方法 synchronizedList(List< T> list)

（3）采用 CopyOnWriteArrayList 容器

9.那你跟我说说 CopyOnWriteArrayList 原理

就是往一个容器添加元素的时候，不直接往当前容器添加，而是先将当前容器进行 Copy，复制出一个新的容器，然后新的容器里添加元素，添加完元素之后，再将原容器的引用指向新的容器。这样做的好处是我们可以对 CopyOnWrite 容器进行并发的读，而不需要加锁，有点读写分离的思想。

10.那你觉得 CopyOnWriteArrayList 会有什么问题

（1）内存占用问题。因为 CopyOnWrite 的写时复制机制，所以在进行写操作的时候，内存里会同时驻扎两个对象的内存。

（2）数据一致性问题，因为还没复制完全的时候，线程读取的还是旧的数据里面的数据，只能保证最终一致性。

11.Hashmap 在多线程情况下会出现什么问题

ConcurrentModificationException（并发修改异常）

put 的时候导致的多线程数据不一致。多线程同时会抢到一个位置，覆盖了数据。在扩容的时候，jdk1.8 之前是采用头插法，当两个线程同时检测到 hashmap 需要扩容，在进行同时扩容的时候有可能会造成链表的死循环。

怎么解决

（1）HashTable 直接 syschronize 锁 put 方法，不推荐

（2）ConcurrentHashMap（推荐）

12. Hashmap 跟 ConcurrentHashMap 区别

（1）Hashmap 允许 key 和 val 为 null，ConcurrentHashMap 不允许

（2）Hashmap 线程不安全，ConcurrentHashMap 线程安全

jdk1.7 对整个数组进行分段（每段都是由若干个 hashEntry 对象组成的链表），每个分段都有一个 Segment 分段锁），每个 Segment 分段锁只会锁住它锁守护的那一段数据，多线程访问不同数据段的数据，不会加锁。ConcurrentHashMap ： jdk1.8ConcurrentHashMap 在 JDK1.8 中采用 Node+CAS+Synchronized 实现线程安全，取消了 segment 分段锁，直接使用 Table 数组存储键值对（与 1.8 中的 HashMap 一样），主要是使用 Synchronized+CAS 的方法来进行并发控制，锁的粒度大大降低了。

13. 那为什么 ConcurrentHashMap 在 JDK1.8 中为什么要使用内置锁 Synchronized 来替换 ReentractLock

重入锁？

Jdk1.6 对 synchronized 进行了优化和升级，引入了锁升级的概念，不再都是重锁的情况

14. ConcurrentHashMap 的 get()方法需要加锁吗？

不需要，get 操作可以无锁是由于 Node 的元素 val 和指针 next 是用 volatile 修饰的，在多线程环境下线程 A 修改结点的 val 或者新增节点的时候是对线程 B 可见的。

15. 假如一张数据量很大的表，百万，千万级别的，如果提高深度分页的查询速度

这个我知道我主要有两种方式可以优化

这个可以走覆盖索引

select * from orders_history where type=8 and

id=(select id from orders_history where type=8 limit 100000,1)，1 秒多

limit 100;先走覆盖索引查出 id，再根据 id 找详情。

如果 id 递增的话，用 between 更快，能达到几十毫秒级别的

select * from orders_history where type=2

and id between 1000000 and 1000100 limit 100;几十毫秒

16. 查找表中多余的重复记录，重复记录是根据单个

字段(Id)来判断

select * from 表 where Id in (select Id from 表 group byId having count(Id) > 1)

17.删除表中多余的重复记录(多个字段)，只留有rowid最小的记录

delete from 表 a where (a.Id,a.seq) in (select Id,seq from 表 group by Id,seq having count() > 1) and rowid not in (select min(rowid) from 表 group by Id,seq having count()>1)

18.查找表中多余的重复记录(多个字段)，不包含 rowid 最小的记录

select * from 表 a where (a.Id,a.seq) in (select Id,seq from 表 group by Id,seq having count() > 1) and rowid not in (select min(rowid) from 表 group by Id,seq having count()>1)

19.假如给一张数据量很大的表加索引，或者加字段，需要怎么操作

肯定不能直接加，如果直接加字段操作会锁表，还可能造成数据库卡死。

我的想法有 2 种方法，

① 创建一个临时的新表，首先复制旧表的结构（包含索引）

② 给新表加上新增的字段

③ 把旧表的数据复制过来

④ 删除旧表，重命名新表的名字为旧表的名字

不过上述这种会造成比较少的数据损失，在第三步可能还有数据写入进旧表

还有一种方法可以在从库进行加字段操作，然后主从切换，这样数据丢失可能性很小

20. 线程池的核心线程数是 new 一个线程池就立马创建吗

这个可以自己设置的，默认情况下是当有任务提交的时候才开始创建，而且就算空闲的线程足以处理新任务，它仍然会创建新的线程去处理，直到核心线程数达到最大值。当然我们可以调用 prestartAllCoreThreads()方法先创建所有核心线程。正常情况下，核心线程池中的线程一但创建了就不会自动被销毁，除非设置了 allowCoreThreadTimeOut=true，或者是线程在执行任务的时候报了异常。

21. 核心线程数能为 0 吗

可以为 0，当核心线程数为 0 的时候，会创建一个非核心线程进行任务处理，其他的跟线程池工作原理一样。

22. 线程池如何判断是否需要回收线程

工作线程启动后，就进入 runWorker(Worker w)方法。

里面是一个 while 循环，线程池会判断 getTask()是否返回 null，如果返回 null 就会被回收。

23. Redis setnx 分布式锁有哪些问题

业务执行时间过长，可能会把其他线程的锁给删了

如何解决：

我使用的 redisson，他有个看门狗机制，会检测程序是否执行完，没执行完会给锁增加过期时间。

24. 那你说说这个看门狗机制的实现原理

当我们在调用 redission 的 lock 方法时没有指定超时时间，就会使用看门狗的默认时间 30 秒，只要抢锁成功的线程，就会开启一个延迟任务，超过看门狗时间的 1/3 就会重新给这个锁设置过期时间。

25. Threadlocal 你了解吗

ThreadLocal 提供线程局部变量。每一个线程在访问 ThreadLocalget 或 set 方法都有自己的变量副本，线程之间数据互不干扰。

是怎么实现的？

当线程请求时，会给每个线程 new 一个自己的 ThreadLocalMap，当执行 set 方法时，值是保存在当前线程的 ThreadLocalMap（跟

hashmap 差不多，是个 entry，保存键值对）里，以当前 threadlocal 对象为 map 的 key，设置的值为 map 的 value，当执行 get 方法中，先通过线程名称获取当前线程的 ThreadLocalMap，然后再通过当前 threadlocal 对象获取自己的 value 属性。

26.聊聊 Threadlocalmap（或者 Threadlocalmap 用了哪种引用，为什么）

底层用了弱引用，其实底层是用了 2 层包装，第一次包装是将 thredlocal 对象变成一个弱引用对象，第二层是定义了一个 entry 对象继承包装后的 threadlocal 对象实现扩展。

27.为什么要使用弱引用（threadlocal 为什么使用完要 remove）

避免内存泄露，因为方法执行完毕后，栈帧中的引用断开，但是此时 Threadlocalmap 里面的 threadlocal 对象还引用着堆里面的实例，造成内存泄露，如果改为弱引用，只要垃圾回收就会回收，降低了内存泄露的风险，remove 可以把 threadlocalmap 的 key 设置成 null，这样 gc 就会回收了。

28.你在项目中有用过 threadlocal 吗？

以前做了一个登录用过，就是业务要区分用户是是登录用户还是临时用户，在拦截器里面创建了一个 threadlocal 对象，然后把用户的

信息对象存进去，比如临时用户给他生成一个标识，然后再传到 controller 进行不同业务逻辑的判断。

29.你项目中用过 syschronize 吗

用过，以前用过一个 OkHttpClient 类，创建这个类的对象时没有用同一个 OkHttpClient 实例并重复使用，每次请求都创建一个新的连接对象，而每个实例都有自己的连接池和线程池，从而导致线程大量堆积，我创建了一个单例模式，然后引入 syschronize，然后双重校验，并且加 volatile 修饰这个单例属性。

```
class GetOkHttpClient {
    public static volatile OkHttpClient instance = null;
    private GetOkHttpClient() {}
    public static OkHttpClient getInstance() {
        if (instance == null) {
            //DCL(双端检索)机制
            synchronized (GetOkHttpClient.class) {
                if (instance == null) {
                    instance = new OkHttpClient();
                }
            }
        }
        return instance;
    }
}
```

30.谈谈Syschronize锁（syschronize是怎么加锁的）

其实每个 java 对象都是由对象头，实例数据和对其填充三部分组成，Syschronize 的锁是存在 java 对象头的 mark word 中，锁升级主要看 mark word 中锁的标志位和释放偏向锁的标志位。

Jdk1.4 后 syschronize 做了什么优化

Jdk1.4 及之前无锁直接上升到重量级锁，1.4 之后引入偏向锁和轻量级锁的概念，让并发性更好。

31.那你解释一下这 4 种锁升级是怎么升级的

（1）无锁:一个线程都还进来，此时是无锁的状态。

（2）偏向锁:当一个线程来访问时，会将 markword 偏向锁标识改成当前线程 id，下次该线程访问只要判断标识位是否是当前线程，如果是则自动获取锁，不需要跟重量级锁一样加锁解锁导致内核态和用户态重复切换，当出现锁竞争时需要释放该锁。

（3）轻量级锁：如果 A 持有偏向锁，B 来抢锁失败了，会一直 cas 抢锁，此时锁升级为轻量级锁，a 线程也会由之前的偏向锁转为轻量级锁，并且还是持有该锁。

（4）重量级锁：当竞争线程较多，并且 cas 自旋长时间抢不到锁会升级为重量级锁。

32.Syschronized 修饰静态和非静态的区别

静态方法锁的是 Class 实例，非静态方法或属性锁的是对象的实例，修饰代码块时锁的是传入的对象，即 this。

33.Redis 5 种数据结构

String、Hash、List、Set、Zset

34.Redis5 种数据结构你都使用了哪些，怎么用的

（1）String 用的最多，比如存登录用户的 token，还有项目中一些键值对的地方

（2）Hash 结构这个也用过，比如文件推送平台，需要区分上游哪个平台传送过来的文件，然后文件还要再一次区分是那种类型的文件。或者针对用户手机唯一 id，大数据会给一个接口先判断用户是属于哪种类型的用户，例如 4 种类型 A,B,C,D，我们平台还需要根据用户所使用的手机类型做进一步区分，比如安卓和 ios，然后再给这个用户推送不同的广告，后台会先配置 A,B,C,D 四种类型的用户，比如少年，青年，中年，老年。这个最为 hash 的 key 区分最外层，然后内层的 map 再根据手机类型进行区分。

（3）Set 结构也用过，比如文件推送过了，我们可以把当天的文件名保存进一个 set 集合，这样通过 set 的 sismember 命令可以快速判断。或者把拉黑的用户 id 存入 set 集合中，用户进入首页直接判断是否在 set 集合中。

（4）Zset 也用过，比如我们平台会给用户 app 上面推广告，后台会配置这个广告的运营时间，然后会把这个广告的点击数存在 zset 当做是分数，公司后台会根据不同的时间投放点击数不同的广告，这个到了时间我就会根据 zset 的 zrangebyscore 取的对应的广告进行投放。

35. 这 5 种数据结构底层是什么数据结构有了解吗

（简单描述，不过多写，粉丝难接受）

（1）String：底层用的是动态字符串，有动态扩容的能力，如果字符

串小于 1M 扩容是字符串 2 倍+1,如果大于 1M，则为扩展后的字符串长度+1M+1,加 1 是结束标识，在字符串长度不一样的时候还会采用不同的编码格式加快查询效率。

（2）List：redis 最早用的是 ziplist（压缩链表），但是当元素个数过多，它的查找效率就会降低。而且如果在 ziplist 里新增或修改数据，ziplist 占用的内存空间还需要重新分配（3.0 版本及之后废弃）。

Redis 先是在 3.0 版本中设计实现了 quicklist。quicklist 结构在 ziplist 基础上，使用链表将 ziplist 串联起来，链表的每个元素就是一个 ziplist。这种设计减少了数据插入时内存空间的重新分配，并且 quicklist 限制了每个节点上 ziplist 的大小，查询效率不会那么低。

Redis 在 5.0 版本中，又实现了 listpack 结构（没有完全替换，不深入介绍，只要达 quicklist 即可）。

（3）Set ：Dict 字典，值为键，value 是 null。

（4）Zset ：当元素个数小于 128 并且每个元素小于 64 字节采用 ziplist 存储，来节省内存。其他情况采用 Dict 字典加跳表的数据结构，DictJI 检查键的唯一性，跳表实现快速排序和查找。

（5）HASH：默认采用 ziplist，ziplist 相连的两个节点保存 key 和 value。当数据量大时，ziplist 会有查询效率问题，会转成 Dict 结构存储。

36. Redis 网络模型了解吗（了解即可，有点难）

我知道有 3 种 select，poll 和 epoll

都使用了 io 多路复用原理。

但是 Select 监听文件（linux 一切东西都是以文件形式表现）有上限，还涉及不少内核拷贝，并且有事件就绪了需要遍历所有的文件找出就绪的事件。

Poll 采用链表存储文件解决了 select 监听 select 监听文件上限的问题，但是有事件发生依然需要遍历整个文件。

Epoll 采用红黑树解决了监听文件上限问题，并且加快了查询就绪事件的效率，并且添加效率也很快。

37. Linux 基本指令

（1）ls：列出所有文件及文件夹。

（2）Pwd：找出当前所在的文件目录。

（3）ps-ef|grep 名称，找出这个应用进程。

（4）rm -rf 递归删除，

（5）cp -rf 递归复制。

（6）Chmod：给文件改权限。

（7）lsof -i:端口号 查看端口是否被占用。

（8）cat 文件名 | grep 关键字，从文件中查找该关键字的记录。

（9）tail -f 文件名查看文件里面的内容，实时打印。

（10）vi：编辑文件

（11）set nu:给文件标识行数。

（12）Linux 创建文件的几种方式：touch 文件名，vi 和 vim，echo。

38. Mysql 默认的隔离级别

可重复读

39. 说下不可重复读和幻读的区别

不可重复读：在并发更新时，另一个事务前后查询数据不一样

幻读：删除或者新增产生数量变化时，另一事务修改或者删除发现影响的行数不一样

40. 知道什么是当前读和快照读吗

最普通的查询语句就是当前读，不加锁

加锁是快照度，比如 insert，delete，update，selet* from *** for update

41. Mysql 是怎么实现可重复读的

（1）快照读：基于 mvcc 多版本并发控制实现的，及 Undo Log +read view 实现的，Undo Log 保存了历史快照，Read View 可见性规则帮助判断当前版本的数据是否可见，当事务执行 SQL 语句前，会得到一个 Read View，可重复读隔离级别下，一个事务里只会获取一次 read view，后面都是共用的，从而保证每次查询的数据都是一样的。

（2）当前读：基于行数加间隙锁实现的。

42. 间隙锁了解吗？

间隙锁主要是在索引记录之间的间隙加锁，从而保证某个间隙内的数

据在锁定情况下不会发生任何变化

43. Mysql 如何排查死锁

可以修改 MySql 系统参数 innodb_print_all_deadlocks 设置成 1，开启状态，这样当发生死锁时，死锁日志会记录到 MySQL 的错误日志文件中。

也可以查看线上的服务器日志

44. Mysql 有哪几种日志文件

我知道的主要有 6 种日志,重做日志（redo log）、回滚日志（undo log）、二进制日志（binlog）、错误日志（errorlog）、慢查询日志（slow query log）、一般查询日志（general log）

45. 都有什么作用

（1）binlog 是 MySQ 服务层维护的一种二进制日志，主要做主从复制、数据恢复和备份；

（2）undo log innerdb 储存引擎层面的日志，提供回滚和多版本并发控制下的读(MVCC)；

（3）redo log 数据备份和数据提交；

（4）errorlog mysql 服务器执行错误时记录进这个错误日志；

（5）slow query log mysql 开启了慢查询，慢 sql 会写入这里；

（6）general log 记录所有的操作日志一般不开启，耗费数据库

性能。

46. redo log 与 binlog 的区别

（1）redo log 是在 InnoDB 存储引擎层产生，而 binlog 是 MySQL 数据库的上层产生的

（2）写入磁盘的时间点不同，binlog 在事务提交完成后进行一次写入。而 redo log 在事务进行中不断地被写入

（3）binlog 在写满或者重启之后，会生成新的 binlog 文件，redo log 是循环使用

47. Mysql 锁你了解几种（都有什么作用）

主要了解 4 种

（1）行锁 锁定一行，锁的是索引，解决当多个线程对数据库进行操作时，会带来数据不一致的情况，会有死锁情况；

（2）表锁 锁定整张表，也是防止解决当多个线程对数据库进行操作时，会带来数据不一致的情况，不会有死锁；

（3）间隙锁 只有在可重复读的情况下才会有可能产生此锁，可以避免幻读，锁的是索引的一段间隔，会有死锁情况；

（4）意向锁，是一种表级锁，与行锁可以同时存在，目的是防止加表锁时需要全表扫描有没有行锁，不会有死锁情况。

48. 什么是索引下推

在 mysql5.6 之前，没有索引下推，比如建立了一个联合索引，先会从第一个索引里面找到合适的数据，再回表查，再过滤，造成多次回表。

而 5.6 之后有引入索引下推，主要直接就可以根据两个联合索引过滤出需要的数据，再回表，减少了回表查的次数。

49. Springmvc 的请求流程（一般 springmvc 就这个问点比较高频）

（1）请求先到前端控制器 DispatcherServlet，如果有过滤器先到过滤器。

（2）然后 DispatcherServlet 会根据请求的 url 找到合适的控制器

（3）然后控制器调用 service 调用 dao 处理业务逻辑

（4）最后会返回一个 ModelAndView

（5）DispatcherServlet 渲染 ModelAndView 最后展示在页面

50. 过滤器，拦截器，DispatcherServlet 执行顺序是什么样的

先到过滤器再到 DispatcherServlet 再到拦截器

51. Spring 三级缓存存的都是什么对象

一级，保存的都是初始化后的 Bean

二级，还没进行属性注入，经过三级缓存处理可能是原对象或代理对）

三级，存放一个对象工厂，和 lambda 表达式，里面保存的都是刚实例化的对象

52. 你在开发中哪些地方用到了反射

文件交换平台，因为对文件操作的步骤配置都不相同，这个我是用了一个枚举，然后枚举第一值是后台配的文件操作步奏具体的 code，第二个值为文件操作类的包名类名，比如 1 代表解压，2 代表转 A 码，3 代码压缩，则后台配置，1，2,3，这样就可以直接通过这个后台配置的 code 找到找到类的包名类名通过反射生成对象进行具体操作。

53. 那你这样写的好处是什么？

提高了程序的灵活性和扩展性，因为再写一个类我只要配置一个枚举就行了，不用在代码里面到处 new 新的对象。

54. Java new 一个对象要做哪些事

（1）在堆区分配对象需要的内存

（2）对所有实例变量赋默认值

（3）执行实例初始化代码

（4）Spring 通过三级缓存解决循环依赖问题（较难，不过多描述，可以看视频理解，尚硅谷周瑜的手写 spring，很详细，还有教你如何处理）

Java 常规题

1. java 语言的三大特性

（1）封装：封装的话主要就是隐藏类内部的细节，提供接口给外部调用就行了，比如像微服务开发，其他部门只要把接口依赖给我们就行了，我们并不知道他们接口内部的细节。

（2）继承：几个类可以继承一个公共类的属性和方法，实现代码的复用，并且子类还可以定义自己的属性和方法。

（3）多态：父类的引用指向子类的对象，并且该引用对象可以表现不同的行为。

2. Java 使用基本数据类型的包装类有什么好处

这个主要是让基本数据类型具有 java 对象相关的属性和方法，可以增加基本数据类型的操作。

还有像 java 集合申明泛型时也必须要求 Object 类型，所以基本数据类型放不进去。

3. java 基本数据类型和包装类型有什么区别

基本数据类型存栈里面，包装类型对象存堆里面，初始值不一样，基本数据类型除了 boolean 是 false，其他的都是 0，包装类型初始值都是 null，像集合里面的泛型都是使用包装类型，没法设置基本数据类型。

基本类型和包装类的区别：

1. 基本类型只能按值传递，包装类按引用类型传递

2. 基本类型在栈中创建，对于对象类型，对象在堆中创建，引用在栈中创建，基本类型由于在栈中，效率会比较高，但是会存在内存泄漏的问题。

4. &和&&的区别

（1）&:逻辑与，a==b&a==c 时，即使 a==b 已经是 false 了，依然还会判断 b 是否等于 c,容易造成空指针异常

（2）&&：短路与，a==b&a==c 时，如果 a==b 是 false 了，那么不会再判断后面的条件是否满足可以避免空指针。

5. ==和 equals 的区别

1、equals()：用来检测两个对象是否相等，即两个对象的内容是否相等。

2、==：用于比较引用和比较基本数据类型时具有不同的功能，具体如下：

（1）基础数据类型：比较的是他们的值是否相等，比如两个 int 类型的变量，比较的是变量的值是否一样。

（2）引用数据类型：比较的是引用的地址是否相同，比如说新建了两个 User 对象，比较的是两个 User 的地址是否一样。

6. Java 中 StringBulider、StringBuffer 有什么区别

StringBuffer：内部加了锁，线程安全；

StringBulider：是非线程安全的，并发高。

7.谈谈 static 关键字

抽象类的抽象方法不能被 static 修饰，因为静态方法不能被重写。静态方法里面不能调用非静态方法，因为非静态方法调用需要创建对象；

static 不能修饰局部变量，因为 static 修饰的变量作用域是全局

8.静态方法能被重写吗，为什么

不能，因为方法重写是在程序运行时选择要执行哪个方法，而 static 修饰的方法是在程序编译时就已经确定要执行哪个方法。

9.重载和重写的区别

都是多态的表现方式

重载是编译时就确定要执行哪个方法了，重写是要在运行时才知道要知道执行哪个方法

重载发生在同一个类，方法名相同，参数类型不同，个数不同或者都不同可认为的方法的重载，而重写发生在父类子类之间，子类方法名和参数列表都要求和父类一样

10.接口和抽象类有什么相同点和区别

相同点：（1）两个都不能被实例化

接口的实现类或抽象类的子类都只有实现了接口或抽象类中的方法后才能实例化。

不同点

（1）jdk1.8 之前接口中全部都要求是抽象方法，java 1.8 中定义 default 方法体，而抽象类可以有非抽象方法。

（2）实现接口的关键字为 implements，抽象类为 extends。一个类可以实现多个接口，但只能继承一个抽象类。

（3）最主要的区别还是接口强调一种功能的实现，而抽象类强调类之间所属关系。

（4）接口成员变量默认为 public static final 而抽象类中成员变量默认 default

11. String 类能被继承吗

这个不能被继承，因为 string 是 final 修饰

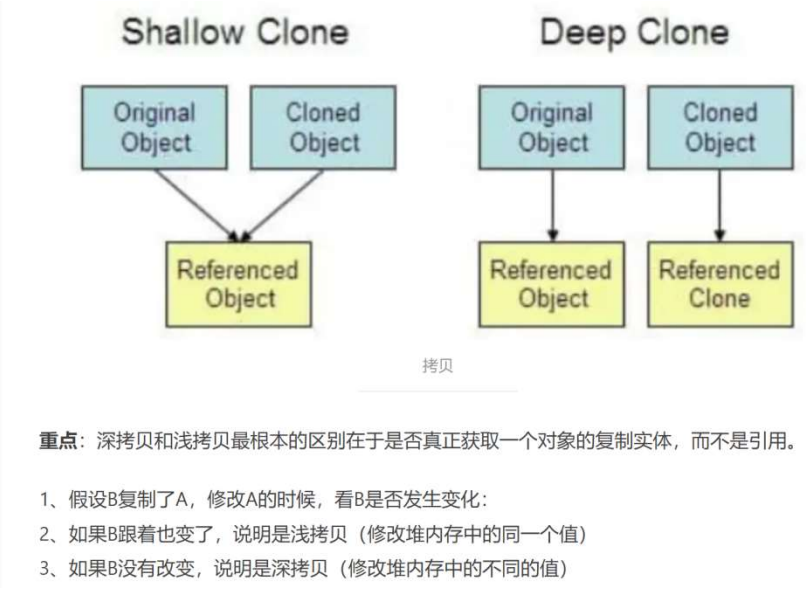
12. 如何自定义一个异常

首先申明一个异常类继承 exception 或者 runtimeException 类，其次编写两个构造方法，一个有参一个无参

13. 深拷贝和浅拷贝的区别

（1）浅拷贝：对基本数据类型进行值传递，对引用数据类型进行引用传递般的拷贝，此为浅拷贝。

（2）深拷贝：对基本数据类型进行值传递，对引用数据类型，创建一个新的对象，并复制其内容，此为深拷贝。



14. comparator 和 comparable 的区别

Comparable 是排序接口，他的泛型只能是自己，Comparator 是比较接口，泛型是任意的

Comparable 接口用于定义对象的自然顺序，是排序接口，Comparator 通常用于定义用户定制的顺序，是比较接口，我们如果需要控制某个类的次序，而该类本身不支持排序（即没有实现 Comparable 接口），那我们就可以建立一个“该类的比较器”来进行排序。

15. 序列化和反序列化

28. Java 序列化,反序列化?

Java 序列化就是指将对象转换为字节序列的过程，反序列化是指将字节序列转换成目标对象的过程。

29.什么情况需要Java序列化?

当 Java 对象需要在网络上传输 或者 持久化存储到文件中时。

30.序列化的实现?

让类实现Serializable接口,标注该类对象是可被序列。

31.如果某些数据不想序列化，如何处理?

在字段面前加 transient 关键字，例如:

```
transient private String phone;//不参与序列化
```

Java 集合（重点 hashmap）

1. ArrayList 和 LinkedList 的区别

（1）ArrayList 底层是动态数组，而 LinkedList 底层是双向链表

（2）ArrayList 随机读取元素较快，可以通过下表直接找到，而 linkedList 需要遍历整个链表长度。删除和插入数据 linkedlist 性能较好，只要断开链表指针重新更改指向即可，而 Arraylist 会涉及底层数组的缩容和扩容。

（3）LinkedList 要占用更多的内存，LinkedList 除了节点存储真实数据还要存储前后节点的位置

2. 如何移除 list 中一个元素

方式一，使用 Iterator ，顺序向下，如果找到元素，则使用 remove 方法进行移除。

方式二，倒序遍历 List ，如果找到元素，则使用 remove 方法进行

移除。

方式三，正序遍历 List ，如果找到元素，则使用 remove 方法进行移除，然后进行索引“自减”。

3.你知道 TreeMap 吗

TreeMap 实现了 NavigableMap 接口，能够对 map 的 key 进行排序。底层是一个数组，但是数组里面的元素是红黑树，数据存在这颗红黑树上面，key 不能重复但是 value 能重复。

4.在日常开发中使用过的 java 集合类

ArrayList，HashMap，HashSet，ConCurentHashmap（多线程组装数据，加快返回速度）

5.为什么 HashMap 的底层数组长度为何总是 2 的 n 次方

主要还是使存入的数据分布均匀，减少冲突。

6.Hashmap put 原理

（1）取 key 的 hashcode 进行高位运算，返回 hash 值

（2）如果 hash 数组为空，直接 resize()

（3）对 hash 进行取模运算计算，得到 key-value 在数组中的存储位置 i

①如果 table[i] == null，直接插入 Node<key,value>

②如果 table[i] != null，判断是否为红黑树。

③如果是红黑树，则判断 TreeNode 是否已存在，如果存在则直接返回 oldnode 并更新；不存在则直接插入红黑树，

④如果是链表，则判断 Node 是否已存在，如果存在则直接返回 oldnode 并更新；不存在则直接插入链表尾部

7.Hashmap 链表超过 8 一定会转为红黑树吗

不是，链表超过 8 且数据总量超过 64 才会转红黑树

8.Jdk1.7 和 1.8 hashmap 有什么区别

（1）在 java1.8 中,如果链表的长度超过了 8,那么链表将转换为红黑树。(数据总量必须大于 64,小于 64 的时候只会扩容)

（2）发生 hash 碰撞时,java1.7 会在链表的头部插入,而 java1.8 会在链表的尾部插入

9.HashMap 和 hashTable 的区别

（1）Hashtable 是线程安全，HashMap 是非线程安全

（2）HashMap 可以使用 null 作为 key 而 Hashtable 则不允许 null 作为 key

（3）HashMap 继承了 AbstractMap，HashTable 继承 Dictionary 抽象类

（4）HashMap 的初始容量为 16，Hashtable 初始容量为 11，两者的填充因子默认都是 0.75。HashMap 扩容时是当前容量翻倍即: capacity *2，Hashtable 扩容时是容量翻倍+1 即:capacity *(2+1)

10. Hashmap 初始容量和负载因子是多少

16 和 0.75

反射

(这个对于年限不高的粉丝非常重要，比较基础，问的多)

1. 你知道创建一个对象有哪几种方式吗

（1）序列化反序列化可以

（2）通过 new 一个对象

（3）通过反射创建也可以

（4）还可以调用一对象的 clone 方法克隆一个对象

2. 什么是反射

反射是在运行状态中，对于任意一个类，能够动态获取类信息以及动态调用对象的方法的功能称为 Java 语言的反射机制。

JAVA 反射机制是在运行状态中，对于任意一个类，都能够知道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方法和属性;这种动态获取的信息以及动态调用对象的方法的功能称为 java 语言的反射机制

3. 获得 class 对象的几种方式

（1）类名.class

（2）对象.getClass。

（3）Class.forName(包名类名)

4. 开发中你觉得哪些地方用到了反射机制

（1）JDBC 中，利用反射动态加载了数据库驱动程序

java Class.forName('com.mysql.jdbc.Driver');//加载 MySQL 的驱动类

（2）Tomcat 服务器中通过反射调用了 Sevlet 的方法

（3）很多框架都用到反射机制，注入属性，调用方法，如 Spring 通过 xml 文件生成 bean 对象并给属性赋值

（4）Eclispe、IDEA 等开发工具利用反射动态刨析对象的类型与结构，动态提示对象的属性和方法

（5）包括我开发中也有过把类型和类包名类名生成一个枚举，然后程序启动通过状态取出包名类名生成对象

5. 那你觉得反射有什么好处

提高了代码灵活性和扩展性，降低耦合性，不用硬编码具体哪个类

6. 如何通过反射创建对象并给属性赋值和调用方法?

（1）先通过反射创建 class 对象

通过一个全限类名创建一个对象

Class.forName(“全限类名”); 例如：com.mysql.jdbc.Driver Driver 类已经被加载到 jvm 中，并且完成了类的初始化工作就行了

类名.class; 获取 Class<？> clz 对象

对象.getClass();

（2）获取构造器对象，通过构造器 new 出一个对象

Clazz.getConstructor([String.class]);

Con.newInstance([参数]);

（3）通过 class 对象创建一个实例对象（就相当于 new 类名（）无参构造器)

Cls.newInstance();

（4）通过 class 对象获得一个属性对象

Field c=cls.getFields()：获得某个类的所有的公共（public）的字段，包括父类中的字段。

Field c=cls.getDeclaredFields()：获得某个类的所有声明的字段，即包括 public、private 和 proteced，但是不包括父类的声明字段

（5）通过 class 对象获得一个方法对象

Cls.getMethod(“方法名”,class……parameaType);（只能获取公共的）

Cls.getDeclareMethod(“方法名”);（获取任意修饰的方法，不能执行私有）

M.setAccessible(true);（让私有的方法可以执行）

让方法执行

IO 和网络编程不多写，这个面试频率不高而且难，优先以其他的为主，学完后可以自己了解一下

多线程(很重要)

1. java 创建线程（Thread）的几种方式

（1）继承于 Thread 类

（2）实现 Runnable 接口

（3）实现 Callable 接口

（4）使用线程池

2. 线程的几种状态

创建，等待，执行，阻塞，死亡

3. 线程 yield 和 join 的区别

yield 是放弃当前 cpu 的执行该线程，但是放弃时间不确定，可能刚刚放弃又获得了 cpu 执行该线程任务。

Join 是把指定线程加入到当前线程，当前线程阻塞，必须指定线程执行完了才会执行当前线程

4. 一个东西的引入必然有缺点和优点，线程池的优缺点？

优点这个我觉得一个是能提高响应速度。还有就是方便管理线程，避免线程一直创建增加服务器压力

缺点的话我觉得引入线程池程序复杂度上去了，肯定要维护这个线程池，还有就是线程池也占用一定资源

5. 线程池的 7 大参数

主要有核心线程数，最大线程池，线程存活时间，时间单位，线程工厂，拒绝策略，阻塞队列

6. 工作原理

首先核心线程数处理任务，如果所有核心线程数还没执行完又来了任务，这个时候任务会先进阻塞队列（这里注意，很多人容易觉得直接会触发最大线程），

如果阻塞队列也满了核心线程数也没空闲线程，这个时候最大线程数的线程会去处理任务，如果最大线程数和核心线程数都没有空闲线程并且阻塞队列也满了，继续来任务就会触发拒绝策略

7. 你们公司用的什么线程池。参数怎么设置的?

我们公司用的是 Threadpoolexecute，参数可自己可配置的这种线程池。这个参数不是我设置的，但是我知道这些参数跟三个指标有关系

1.任务量、2.单个任务的处理时间、3 页面允许响应的最大时间，这三个指标有一个复杂的运算。

这个 up 说明一下，假如你用 excel 导入 1000 条数据。任务量就是 1000，单个任务的处理时间就是一条数据插入数据库的时间，允许的响应时间就是前端多久给提示，比如导入成功这种的。

8. 你还知道哪几种线程池?

可缓存线程池，可定长度线程池，可定时线程池，单例线程池

9. 那开发中能不能直接使用这种线程池?

一般不用，这些线程池参数配置死了，不可控。

10. 拒绝策略有哪几种?

（1）直接丢弃任务，抛异常。

（2）不丢弃，用当前线程去执行任务。

（3）忽略，不抛异常。

（4）移除最早进入队列的任务。

（5）还可以实现一个拒绝策略处理器接口自己定义拒绝策略。

11. 你们公司用的什么拒绝策略?

我们公司用的是不拒绝，因为我们公司业务这个任务数据不能丢失。

12. 那你举个例子说一下你的线程池使用场景和流程

（1）因为我们公司有一个 excel 导入数据的过程，大概在 5000 条导入时间就会很长，单线程执行可能都要 10 几秒页面才会返回，
我是分批进行导入的，首先读出 excel 的 5000 条数据，然后存入一个集合中，当时也验证了，大概在一批 200 条开启一个线程,最合适。
首先用总数量除以 200 算出开启线程数量，然后 for 循环里面用 list

的 sublist 方法，一次循环取 200 条给线程去插入数据库，

最后 5000 条数据大概 1 秒左右就返回了。

（2）还有个场景就是打开一个页面加载速度太慢了，因为页面取得数据来自很多表和其他部门的接口，这个也用多线程去异步查询然后组装了。

13. 多线程 execute 和 submit 的区别

（1）execute 只能提交 Runnable 类型的任务，无返回值。submit 既可以提交 Runnable 类型的任务，也可以提交 Callable 类型的任务，会有一个类型为 Future 的返回值，但当任务类型为 Runnable 时，返回值为 null。

（2）execute 在执行任务时，如果遇到异常会直接抛出，而 submit 不会直接抛出，只有在使用 Future 的 get 方法获取返回值时，才会获取异常

14. shutDown 和 shutdownNow 的区别

（1）shutDown()

当线程池调用该方法时,不能再往线程池中添加任何任务，否则将会抛出 RejectedExecution 异常。但是，此时线程池不会立刻退出，直到添加到线程池中的任务都已经处理完成，才会退出，包括队列里面的任务。

（2）shutdownNow()

执行该方法，线程池的状态立刻变成 STOP 状态，会试图停止所有正在执行的线程，不再处理队列中等待的任务 但是正在执行的任务都执行完成了才能退出。

15.线程池的核心线程数怎么设置

（1）CPU 密集型是 CPU 核数+1，+1 是因为 CPU 密集型一般也有一些 IO 操作，所以再加一个线程来把等待 IO 的 CPU 时间利用起来；

（2）IO 密集型是 2 倍的 cpu 核数，是因为 IO 操作 CPU 使用率并不高，因此可以让 CPU 在等待 IO 的时候去处理别的任务，充分利用 CPU 时间。

JUC 包面试题

（这个不写太多，了解即可，太耗时间）

1.Lock 和 Synchronizaed 的区别

（1）Lock（接口）是手动加锁、解锁，因此可能会出现死锁。可以设置公平或非公平，如果一个线程获取锁，其他线程可以不用等待。（tryLock()方法）。

（2）Synchronizaed（关键字）自动解锁，不会死锁。是非公平锁，如果一个线程获得锁，其他线程会一直等待。

底层实现不一样：

（1）synchronized: 底层使用指令码方式来控制锁的，映射成字节码

指令就是增加来两个指令：monitorenter 和 monitorexit。当线程执行遇到 monitorenter 指令时会尝试获取内置锁，如果获取锁则锁计数器+1，如果没有获取锁则阻塞；当遇到 monitorexit 指令时锁计数器-1，如果计数器为 0 则释放锁。

（2）Lock 底层是 CAS+aqs+一个 volatile 关键字修饰的状态，实现的加锁解锁

2. 可重入锁了解吗？

可重入锁的意思就是线程获取锁之后可以不用释放锁再次获取锁。

3. 举一个 java 死锁的例子

```java
private static String A="A";
private static String B="B";
public static void main(String[] args){
    new DeadLockDemo().deadLock();
}
private void deadLock(){
    Thread threadA=new Thread(new Runnable(){
        @Override
        public void run(){
            synchronized(A){
                try {
                    Thread.currentThread().sleep(2000);  //睡两秒确保了阻塞，不然可能直接运行下去了
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                synchronized(B){
                    System.out.println("AB");
                }
            }
        }
    });
    Thread threadB=new Thread(new Runnable(){
        @Override
        public void run(){
            synchronized(B){
                try {
                    Thread.currentThread().sleep(2000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                synchronized(A){
                    System.out.println("BA");
                }
            }
        }
    });
    threadA.start();
```

4. Java 如何定位一个死锁

（1）jps 指令查看项目的 pid(进程 id)

（2）然后使用 jstack -l pid 查看死锁信息

（3）通过打印信息我们可以找到发生死锁的代码是在哪个位置

5. Lock 锁公平和非公平锁有什么区别

非公平锁在调用 lock 后，首先就会调用 CAS 进行一次抢锁，如果这个时候恰巧锁没有被占用，那么直接就获取到锁返回了。但是

公平锁会判断aqs队列是否有线程处于等待状态，如果有则不去抢锁，直接排到队列后面等待。

6.你有用过 juc 包下面哪些类

ConcurenthashMap、CountdownLarch、AtomicBoolean、线程池

ConcurentHashMap(面试高频)

1.ConcurrentHashMap 在 JDK1.7 和 JDK1.8 区别

（1）加锁的地方不一样，jdk1.8 的实现降低锁的粒度，jdk1.7 锁的粒度是基于段的，每个段包含多个 HashEntry，而 jdk1.8 锁的粒度就是 Node。

（2）数据结构不一样：jdk1.7：Segment+HashEntry。jdk1.8：数组+链表+红黑树+CAS+synchronized。

2.ConcurrentHashmap 的key，value是否可以为null

不行。如果 key 或者 value 为 null 会抛出空指针异常。

因为 ConcurrentHashMap 是线程安全的，在并发环境下，如果调用 get 方法获取到 null 之后，有可能是 key 不存在也有可能是 key 存在，但是 value 为 null

hashMap 是单线程所以能用 containsKey 方法判断有没有该 key。

但是如果并发下 concurentmap 如果调用 containsKey 之前没有这个 key，期望返回 false，但是调用之前可能有线程插入了这个 key，结果

返回 tue 了。

Redis 面试题

1.Redis 为什么快

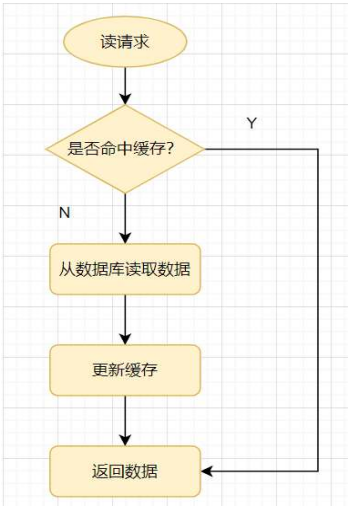
（1）Redis 使用了单线程架构+IO 多路复用模型

（2）纯内存访问

（3）单线程避免了上下文切换带来的资源消耗

2.什么是缓存击穿、缓存穿透、缓存雪崩

常见的缓存使用方式：读请求来了，先查下缓存，缓存有值命中，就直接返回，缓存没命中，就去查数据库，然后把数据库的值更新到缓存，再返回。



（1）缓存穿透

指查询一个一定不存在的数据，由于缓存是不命中时需要从数据库查询，查不到数据则不写入缓存，这将导致这个不存在的数据每次

请求都要到数据库去查询，进而给数据库带来压力。

如何避免缓存穿透？

①如果是非法请求，我们在 API 入口，对参数进行校验，过滤非法值。

②如果查询数据库为空，我们可以给缓存设置个空值，或者默认值。

③使用布隆过滤器快速判断数据是否存在。即一个查询请求过来时，先通过布隆过滤器判断值是否存在，存在才继续往下查。

（2）缓存雪崩

指同一个时刻缓存中数据大批量到过期时间失效，而查询数据量巨大，请求都直接访问数据库，引起数据库压力过大甚至 down 机。

如何避免缓存雪崩？

不要把所有 key 过期时间设置为一样的，让过期时间相对离散一点。如采用一个较大固定值+一个较小的随机值，5 小时+0 到 1800 秒这样。

（3）缓存击穿

一个比较热门的键过期了，多个请求访问这个键，因为键不存在了，这将导致每次访问这个不存在的数据都是直接请求到数据库，增加数据库压力。

如何避免缓存击穿呢？

①设置热门键不过期

②考虑加分布式锁，获取锁的第一个线程把数据库数据拉出来加进缓

存，后续的请求就不用去数据库查询了

3.redis 字符串最大不能超过多少

512M

4.Redis 默认分多少个数据库

16 个

5.Redis 过期策略

（1）定时过期：每个设置过期时间的 key 都需要创建一个定时器，到过期时间就会立即对 key 进行清除。

（2）惰性过期：只有当访问一个 key 时，才会判断该 key 是否已过期，过期则清除。

（3）定期过期：每隔一定的时间，会扫描一定数量的数据库的 expires 字典中一定数量的 key，并清除其中已过期的 key。

6.Redis 的持久化机制有哪些？优缺点说说

（1）RDB

就是把内存数据以快照的形式保存到磁盘上。RDB 持久化，是指在指定的时间间隔内，执行指定次数的写操作，将内存中的数据集快照写入磁盘中，它是 Redis 默认的持久化方式。执行完操作后，在指定目录下会生成一个 dump.rdb 文件，Redis 重启的时候，通过加载

dump.rdb 文件来恢复数据。

RDB 的优点：适合大规模的数据恢复场景，如备份，全量复制等

RDB 缺点：没办法做到实时持久化/秒级持久化。

（2）AOF

采用日志的形式来记录每个写操作，追加到文件中，重启时再重新执行 AOF 文件中的命令来恢复数据。它主要解决数据持久化的实时性问题。默认是不开启的。

AOF 的优点：数据的一致性和完整性更高

AOF 的缺点：AOF 记录的内容越多，文件越大，数据恢复变慢。

7.MySQL 与 Redis 如何保证双写一致性

（1）缓存延时双删：先删除缓存，再更新数据库，休眠一会（比如 1 秒），再次删除缓存。

（2）删除缓存重试机制：写请求更新数据库缓存因为某些原因，删除失败，把删除失败的 key 放到消息队列，消费消息队列的消息，获取要删除的 key 重试删除缓存操作

（3）读取 biglog 异步删除缓存：以 mysql 为例，可以使用阿里的 canal 将 binlog 日志采集发送到 MQ 队列里面，然后通过 ACK 机制确认处理这条更新消息，删除缓存，保证数据缓存一致性

8.Redis 内存溢出策略

1. 当内存使用超过配置的时候会返回错误，不会删除任何键

2. 删除最久没有使用的键

3. 从所有 key 随机删除

4. 从过期键的集合中随机驱逐

5. 从所有键中驱逐使用频率最少的键 Mysql

6. 从设置了过期时间的键集合中驱逐最久没有使用的键

7. 从配置了过期时间的键中驱逐马上就要过期的键

8. 从所有配置了过期时间的键中驱逐使用频率最少的键

Mysql 面试题

1.CHAR 和 VARCHAR 区别？

（1）char 表示定长，长度固定，varchar 表示变长，长度可变，char 如果插入的长度小于定义长度时，用空格填充，varchar 小于定义长度时，还是按实际长度存储。

（2）存储的容量不同，对 char 来说，最多能存放字符个数 255，和编码无关，对 varchar 来说，最多存放 65532 个字符。

2.索引哪些情况会失效？（记 5 种即可）

查询条件包含 or，可能导致索引失效
如何字段类型是字符串，where 时一定用引号括起来，否则索引失效
like 通配符可能导致索引失效。
联合索引，查询时的条件列不是联合索引中的第一个列，索引失效。
在索引列上使用 mysql 的内置函数，索引失效。
对索引列运算（如，+、-、*、/），索引失效。
索引字段上使用（！= 或者 < >，not in）时，可能会导致索引失效。
索引字段上使用 is null， is not null，可能导致索引失效。
左连接查询或者右连接查询查询关联的字段编码格式不一样，可能导致索引失效。
mysql 估计使用全表扫描要比使用索引快,则不使用索引。

3. 索引不适合哪些场景（哪些字段不适合建索引）

数据量少的不适合加索引
更新比较频繁的也不适合加索引
区分度低的字段不适合加索引（如性别）

4. 索引的一些定义

- 覆盖索引　查询的字段刚好满足建立索引的字段，不用回表
- 回表　　查询的字段在索引列中可能没有，需要通过主键 id 回主键索引表中查询出需要的数据
- 最左前缀原则　指在复合索引（多列索引）中，查询条件应该与索引中的列顺序相匹配，从左到右进行过滤
- 索引下推　使用一张用户表 tuser，表里创建联合索引（name, age），select * from tuser where name like '张%' and age=10;在 MySQL 5.6 之前，存储引擎根据通过联合索引找到 name likelike '张%' 的主键 id，比如有 10 条，逐一进行回表扫描，回表了 10 次，去聚簇索引找到完整的行记录，server 层再对数据根据 age=10 进行筛选，筛选后假如就只有一条。

  而 MySQL 5.6 以后， 存储引擎根据（name，age）联合索引，找到，由于联合索引中包含列，所以存储引擎直接再联合索引里按照 age=10 过滤。所以只回表了一次

5. MySQL 如何定位死锁问题

- 查看死锁日志 show engine innodb status;
- 找出死锁 Sql

6. 日常工作中你是怎么优化 SQL 的？

- 加索引
- 避免返回不必要的数据
- 适当分批量进行
- 优化 sql 结构
- 分库分表
- 读写分离

7. 说说你对分库与分表的理解

- 水平分库：以字段为依据，按照一定策略（hash、range 等），将一个库中的数据拆分到多个库中。

- 水平分表：以字段为依据，按照一定策略（hash、range 等），将一个表中的数据拆分到多个表中。
- 垂直分库：以表为依据，按照业务归属不同，将不同的表拆分到不同的库中。
- 垂直分表：以字段为依据，按照字段的活跃性，将表中字段拆到不同的表（主表和扩展表）中。

8. 分库与分表后可能有哪些问题

- 事务问题：需要用分布式事务啦
- 跨节点 Join 的问题：解决这一问题可以分两次查询实现
- 跨节点的 count,order by,group by 以及聚合函数问题：分别在各个节点上得到结果后在应用程序端进行合并。
- 数据迁移，容量规划，扩容等问题
- ID 问题：数据库被切分后，不能再依赖数据库自身的主键生成机制啦，最简单可以考虑 UUID
- 跨分片的排序分页问题

9. InnoDB 与 MyISAM 的区别（记 5 点即可）

- InnoDB 支持事务，MyISAM 不支持事务
- InnoDB 支持外键，MyISAM 不支持外键
- InnoDB 支持 MVCC(多版本并发控制)，MyISAM 不支持
- select count(*) from table 时，MyISAM 更快，因为它有一个变量保存了整个表的总行数，可以直接读取，InnoDB 就需要全表扫描。
- Innodb 不支持全文索引，而 MyISAM 支持全文索引（5.7 以后的 InnoDB 也支持全文索引）
- InnoDB 支持表、行级锁，而 MyISAM 支持表级锁。
- InnoDB 表必须有主键，而 MyISAM 可以没有主键
- Innodb 表需要更多的内存和存储，而 MyISAM 可被压缩，存储空间较小，。
- Innodb 按主键大小有序插入，MyISAM 记录插入顺序是，按记录插入顺序保存。
- InnoDB 存储引擎提供了具有提交、回滚、崩溃恢复能力的事务安全，与 MyISAM 比 InnoDB 写的效率差一些，并且会占用更多的磁盘空间以保留数据和索引

10. 数据库索引的原理，为什么要用 B+树，为什么不用二叉树

查询是否够快，效率是否稳定，存储数据多少，以及查找磁盘次数

11. 为什么不是一般二叉树

如果二叉树特殊化为一个链表，相当于全表扫描。平衡二叉树相比于二叉查找树来说，查找效率更稳定，总体的查找速度也更快。

12. 为什么不是平衡二叉树

如果树这种数据结构作为索引，那我们每查找一次数据就需要从磁盘中读取一个节点，但是平衡二叉树可是每个节点只存储一个键值和数据的，如果是 B 树，可以存储更多的节点数据，树的高度也会降低，因此读取磁盘的次数就减少

13. 为什么不是 B 树而是 B+树

1）B+树非叶子节点上是不存储数据的，仅存储键值，而 B 树节点中不仅存储键值，也会存储数据。innodb 中页的默认大小是 16KB，如果不存储数据，那么就会存储更多的键值，相应的树的阶数（节点的子节点树）就会更大，树就会更矮更胖，如此一来我们查找数据进行磁盘的 IO 次数有会再次减少，数据查询的效率也会更快。

2）B+树索引的所有数据均存储在叶子节点，而且数据是按照顺序排列的，链表连着的。那么 B+树使得范围查找，排序查找，分组查找以及去重查找变得异常简单。

14. 聚集索引与非聚集索引的区别

- 一个表中只能拥有一个聚集索引，而非聚集索引一个表可以存在多个。
- 聚集索引，索引中键值的逻辑顺序决定了表中相应行的物理顺序；非聚集索引，索引中索引的逻辑顺序与磁盘上行的物理存储顺序不同。
- 索引是通过二叉树的数据结构来描述的，我们可以这么理解聚簇索引：索引的叶节点就是数据节点。而非聚簇索引的叶节点仍然是索引节点，只不过有一个指针指向对应的数据块。
- 聚集索引：物理存储按照索引排序；非聚集索引：物理存储不按照索引排序；

15. limit 1000000 查找后面的 10 条数据 加载很慢的话，你是怎么解决的呢？

方案一：如果 id 是连续的，可以这样，返回上次查询的最大记录(偏移量)，再往下 limit
select id，name from employee where id>1000000 limit 10.
方案二：在业务允许的情况下限制页数：
建议跟业务讨论，有没有必要查这么后的分页啦。因为绝大多数用户都不会往后翻太多页。

方案三：order by + 索引（id 为索引）
select id，name from employee order by id limit 1000000，10
方案四：利用延迟关联或者子查询优化超多分页场景。（先快速定位需要获取的 id 段，然后再关联）
SELECT a.* FROM employee a, (select id from employee where 条件 LIMIT 1000000,10) b where a.id=b.id

16. 知道哪些分布式主键？

- 数据库自增长序列或字段。
- UUID。
- Redis 生成 ID
- Twitter 的 snowflake 算法
- 利用 zookeeper 生成唯一 ID

17. 事务的隔离级别有哪些？MySQL 的默认隔离级别是什么

- 读未提交（Read Uncommitted）
- 读已提交（Read Committed）
- 可重复读（Repeatable Read）
- 串行化（Serializable）

Mysql 默认的事务隔离级别是可重复读(Repeatable Read)

18. 什么是幻读，脏读，不可重复读呢？

- 事务 A、B 交替执行，事务 A 被事务 B 干扰到了，因为事务 A 读取到事务 B 未提交的数据，这就是**「脏读」**
- 在一个事务范围内，两个相同的查询，读取同一条记录，却返回了不同的数据，这就是**「不可重复读」**。
- 事务 A 查询一个范围的结果集，另一个并发事务 B 往这个范围中插入/删除了数据，并静悄悄地提交，然后事务 A 再次查询相同的范围，两次读取得到的结果集不一样了，这就是**「幻读」**。

19. 在高并发情况下，如何做到安全的修改同一行数据？

使用悲观锁

悲观锁思想就是，当前线程要进来修改数据时，别的线程都得拒之门外~ 比如，可以使用 select…for update ~

```
select * from User where name=‘jay’ for update
```

以上这条 sql 语句会锁定了 User 表中所有符合检索条件（name=‘jay’）的记录。本次事务提交之前，别的线程都无法修改这些记录。

使用乐观锁

乐观锁思想就是，有线程过来，先放过去修改，如果看到别的线程没修改过，就可以修改成功，如果别的线程修改过，就修改失败或者重试。实现方式：乐观锁一般会使用版本号机制或 CAS 算法实现。

20. MySQL 事务得四大特性以及实现原理

- 原子性： 事务作为一个整体被执行，包含在其中的对数据库的操作要么全部被执行，要么都不执行。
- 一致性： 指在事务开始之前和事务结束以后，数据不会被破坏，假如 A 账户给 B 账户转 10 块钱，不管成功与否，A 和 B 的总金额是不变的。
- 隔离性： 多个事务并发访问时，事务之间是相互隔离的，即一个事务不影响其它事务运行效果。简言之，就是事务之间是进水不犯河水的。
- 持久性： 表示事务完成以后，该事务对数据库所作的操作更改，将持久地保存在数据库之中。

「事务 ACID 特性的实现思想」

- 原子性：是使用 undo log 来实现的，如果事务执行过程中出错或者用户执行了 rollback，系统通过 undo log 日志返回事务开始的状态。
- 持久性：使用 redo log 来实现，只要 redo log 日志持久化了，当系统崩溃，即可通过 redo log 把数据恢复。
- 隔离性：通过锁以及 MVCC,使事务相互隔离开。
- 一致性：通过回滚、恢复，以及并发情况下的隔离性，从而实现一致性。

21. 如果某个表有近千万数据，CRUD 比较慢，如何优化

- 分表方案（水平分表，垂直分表，切分规则 hash 等）
- 分库分表中间件（Mycat，sharding-jdbc 等）
- 分库分表一些问题（事务问题？跨节点 Join 的问题）
- 解决方案（分布式事务等）

22. 一条 SQL 语句在 MySQL 中如何执行的

- 先检查该语句是否有权限
- 如果没有权限，直接返回错误信息
- 如果有权限，在 MySQL8.0 版本以前，会先查询缓存。
- 如果没有缓存，分析器进行词法分析，提取 sql 语句 select 等的关键元素。然后判断 sql 语句是否有语法错误，比如关键词是否正确等等。
- 优化器进行确定执行方案
- 进行权限校验，如果没有权限就直接返回错误信息，如果有权限就会调用数据库引擎接口，返回执行结果。

23. 一条 sql 执行过长的时间，你如何优化，

从哪些方面入手

- 查看是否涉及多表和子查询，优化 Sql 结构，如去除冗余字段，是否可拆表等
- 优化索引结构，看是否可以适当添加索引
- 数量大的表，可以考虑进行分离/分表（如交易流水表）
- 数据库主从分离，读写分离
- explain 分析 sql 语句，查看执行计划，优化 sql
- 查看 mysql 执行日志，分析是否有其他方面的问题

24. Blob 和 text 有什么区别？

- Blob 用于存储二进制数据，而 Text 用于存储大字符串。
- Blob 值被视为二进制字符串（字节字符串）,它们没有字符集，并且排序和比较基于列值中的字节的数值。
- text 值被视为非二进制字符串（字符字符串）。它们有一个字符集，并根据字符集的排序规则对值进行排序和比较。

25. Mysql 中有哪几种锁

- 表锁： 开销小，加锁快；锁定力度大，发生锁冲突概率高，并发度最低;不会出现死锁。
- 行锁： 开销大，加锁慢；会出现死锁；锁定粒度小，发生锁冲突的概率低，并发度高。
- 页锁： 开销和加锁速度介于表锁和行锁之间；会出现死锁；锁定粒度介于表锁和行锁之间，并发度一般

26. Hash 索引和 B+树区别是什么？你在设计索引是怎么抉择的？

- B+树可以进行范围查询，Hash 索引不能。
- B+树支持联合索引的最左侧原则，Hash 索引不支持。
- B+树支持 order by 排序，Hash 索引不支持。
- Hash 索引在等值查询上比 B+树效率更高。
- B+树使用 like 进行模糊查询的时候，like 后面（比如%开头）的话可以起到优化的作用，Hash 索引根本无法进行模糊查询。

27. mysql 的内连接、左连接、右连接有什么区别？

- Inner join 内连接，在两张表进行连接查询时，只保留两张表中完全匹配的结果集
- left join 在两张表进行连接查询时，会返回左表所有的行，即使在右表中没有匹配的记录。
- right join 在两张表进行连接查询时，会返回右表所有的行，即使在左表中没有匹配的记录。

28. 说说 MySQL 的基础架构图

- 第一层负责连接处理，授权认证，安全等等
- 第二层负责编译并优化 SQL
- 第三层是存储引擎。

29. 说一下数据库的三大范式

- 第一范式：数据表中的每一列（每个字段）都不可以再拆分。
- 第二范式：在第一范式的基础上，分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：在满足第二范式的基础上，表中的非主键只依赖于主键，而不依赖于其他非主键。

30. 索引有哪几种类型？

- 主键索引: 数据列不允许重复，不允许为 NULL，一个表只能有一个主键。
- 唯一索引: 数据列不允许重复，允许为 NULL 值，一个表允许多个列创建唯一索引。
- 普通索引: 基本的索引类型，没有唯一性的限制，允许为 NULL 值。
- 全文索引：是目前搜索引擎使用的一种关键技术，对文本的内容进行分词、搜索。
- 覆盖索引：查询列要被所建的索引覆盖，不必读取数据行
- 组合索引：多列值组成一个索引，用于组合搜索，效率大于索引合并

31. 创建索引的三种方式

- 在执行 CREATE TABLE 时创建索引
- CREATE TABLE `employee` (
- `id` int(11) NOT NULL,
- `name` varchar(255) DEFAULT NULL,
- `age` int(11) DEFAULT NULL,
- `date` datetime DEFAULT NULL,
- `sex` int(1) DEFAULT NULL,
- PRIMARY KEY (`id`),
- KEY `idx_name` (`name`) USING BTREE
-) ENGINE=InnoDB DEFAULT CHARSET=utf8;
- 使用 ALTER TABLE 命令添加索引
- ALTER TABLE table_name ADD INDEX index_name (column);
- 使用 CREATE INDEX 命令创建
- CREATE INDEX index_name ON table_name (column);

32. 百万级别或以上的数据，你是如何删除

的？

- 我们想要删除百万数据的时候可以先删除索引
- 然后批量删除其中无用数据
- 删除完成后重新创建索引。

33. 非聚簇索引一定会回表查询吗？

- 不一定，如果查询语句的字段全部命中了索引，那么就不必再进行回表查询（覆盖索引就是这么回事）。

34. 什么是死锁？怎么解决？

死锁是指两个或多个事务在同一资源上相互占用，并请求锁定对方的资源，从而导致恶性循环的现象。

- 解决：如果不同程序会并发存取多个表，尽量约定以相同的顺序访问表，可以大大降低死锁机会。
- 在同一个事务中，尽可能做到一次锁定所需要的所有资源，减少死锁产生概率；
- 对于非常容易产生死锁的业务部分，可以尝试使用升级锁定颗粒度，通过表级锁定来减少死锁产生的概率；
- 如果业务处理不好可以用分布式事务锁或者使用乐观锁
- 死锁与索引密不可分，解决索引问题，需要合理优化你的索引，

”

35. UNION 与 UNION ALL 的区别？

- Union：对两个结果集进行并集操作，不包括重复行，同时进行默认规则的排序；
- Union All：对两个结果集进行并集操作，包括重复行，不进行排序；
- UNION 的效率高于 UNION ALL

36. 字段为什么要求定义为 not null？

null 值会占用更多的字节，

37. MySQL 数据库 cpu 飙升的话，要怎么处理呢

「排查过程：」

- 使用 top 命令观察，确定是 mysqld 导致还是其他原因。
- 如果是 mysqld 导致的，show processlist，查看 session 情况，确定是不是有消耗资源的 sql 在运行。
- 找出消耗高的 sql，看看执行计划是否准确， 索引是否缺失，数据量是否太大。

「处理：」

- kill 掉这些线程(同时观察 cpu 使用率是否下降)，
- 进行相应的调整(比如说加索引、改 sql、改内存参数)
- 重新跑这些 SQL。

「其他情况：」

也有可能是每个 sql 消耗资源并不多，但是突然之间，有大量的 session 连进来导致 cpu 飙升，这种情况就需要跟应用一起来分析为何连接数会激增，再做出相应的调整，比如说限制连接数等

38. MySQL 中 DATETIME 和 TIMESTAMP 的区别

存储精度都为秒

区别：

- DATETIME 的日期范围是 1001——9999 年；TIMESTAMP 的时间范围是 1970——2038 年
- DATETIME 存储时间与时区无关；TIMESTAMP 存储时间与时区有关，显示的值也依赖于时区
- DATETIME 的存储空间为 8 字节；TIMESTAMP 的存储空间为 4 字节
- DATETIME 的默认值为 null；TIMESTAMP 的字段默认不为空(not null)，默认值为当前时间(CURRENT_TIMESTAMP)

39. count(1)、count(*) 与 count(列名) 的区别？

- count(*)包括了所有的列，相当于行数，在统计结果的时候，不会忽略列值为 NULL
- count(1)包括了忽略所有列，用 1 代表代码行，在统计结果的时候，不会忽略列值为 NULL
- count(列名)只包括列名那一列，在统计结果的时候，会忽略列值为空（这里的空不是只空字符串或者 0，而是表示 null）的计数，即某个字段值为 NULL 时，不统计。

40. Sql 优化流程

可以通过 EXPLAIN 或者 DESC 命令获取 MySQL 如何执行 SELECT 语句的信息，包括在 SELECT 语句执行过程中表如何连接和连接的顺序。

id	select 查询的序列号，是一组数字，表示的是查询中执行 select 子句或者是操作表的顺序。相同的 id 依次从上到下执行，id 越大的先执行

type 表示表的连接类型，性能由好到差的连接类型为(system —> const -----> eq_ref ------> ref -------> ref_or_null----> index_merge —> index_subquery -----> range -----> index ------> all)
possible_keys 表示查询时，可能使用的索引
key 表示实际使用的索引
Extra 执行情况的说明和描述
Rows 扫描行的数量

Spring 面试题

1.Spring 用到了哪些设计模式?

（1）简单工厂模式：BeanFactory 就是简单工厂模式的体现，根据传入一个唯一标识来获得 Bean 对象。

（2）工厂方法模式：FactoryBean 就是典型的工厂方法模式。spring 在使用 getBean()调用获得该 bean 时，会自动调用该 bean 的 getObject()方法。每个 Bean 都会对应一个 FactoryBean，如 SqlSessionFactory 对

应 SqlSessionFactoryBean。

（3）单例模式：一个类仅有一个实例，提供一个访问它的全局访问点。Spring 创建 Bean 实例默认是单例的。

（4）适配器模式：SpringMVC 中的适配器 HandlerAdatper。由于应用会有多个 Controller 实现，如果需要直接调用 Controller 方法，那么需要先判断是由哪一个 Controller 处理请求，然后调用相应的方法。当增加新的 Controller，需要修改原来的逻辑，违反了开闭原则（对修改关闭，对扩展开放）。

（5）代理模式：Spring 的 aop 使用了动态代理，有两种方式 JdkDynamicAopProxy 和 Cglib2AopProxy。

（6）观察者模式：Spring 中 observer 模式常用的地方是 listener 的实现，如 ApplicationListener。

（7）模板模式： Spring 中 jdbcTemplate、hibernateTemplate 等，就使用到了模板模式。

2.什么是 AOP？

面向切面编程，作为面向对象的一种补充，将公共逻辑（事务管理、日志、缓存等）封装成切面，跟业务代码进行分离，可以减少系统的重复代码和降低模块之间的耦合度。切面就是那些与业务无关，但所有业务模块都会调用的公共逻辑。

3.AOP 有哪些实现方式？

AOP 有两种实现方式：静态代理和动态代理。

（1）静态代理

代理类在编译阶段生成，在编译阶段将通知织入 Java 字节码中，也称编译时增强。AspectJ 使用的是静态代理。

缺点：代理对象需要与目标对象实现一样的接口，并且实现接口的方法，会有冗余代码。同时，一旦接口增加方法，目标对象与代理对象都要维护。

（2）动态代理

代理类在程序运行时创建，AOP 框架不会去修改字节码，而是在内存中临时生成一个代理对象，在运行期间对业务方法进行增强，不会生成新类。

4. JDK 动态代理和 CGLIB 动态代理的区别？

（1）JDK 动态代理

如果目标类实现了接口，Spring AOP 会选择使用 JDK 动态代理目标类。代理类根据目标类实现的接口动态生成，不需要自己编写，生成的动态代理类和目标类都实现相同的接口。JDK 动态代理的核心是 InvocationHandler 接口和 Proxy 类。

缺点：目标类必须有实现的接口。如果某个类没有实现接口，那么这个类就不能用 JDK 动态代理。

（2）CGLIB 动态代理

通过继承实现。如果目标类没有实现接口，那么 Spring AOP 会选

择使用 CGLIB 来动态代理目标类。CGLIB（Code Generation Library）可以在运行时动态生成类的字节码，动态创建目标类的子类对象，在子类对象中增强目标类。

CGLIB 是通过继承的方式做的动态代理，因此如果某个类被标记为 final，那么它是无法使用 CGLIB 做动态代理的。

优点：目标类不需要实现特定的接口，更加灵活。

（3）什么时候采用哪种动态代理？

①如果目标对象实现了接口，默认情况下会采用 JDK 的动态代理实现 AOP

②如果目标对象实现了接口，可以强制使用 CGLIB 实现 AOP

③如果目标对象没有实现了接口，必须采用 CGLIB 库

（4）两者的区别：

jdk 动态代理使用 jdk 中的类 Proxy 来创建代理对象，它使用反射技术来实现，不需要导入其他依赖。cglib 需要引入相关依赖：asm.jar，它使用字节码增强技术来实现。

当目标类实现了接口的时候 Spring Aop 默认使用 jdk 动态代理方式来增强方法，没有实现接口的时候使用 cglib 动态代理方式增强方法。

5.什么是 IOC？

控制反转，由 Spring 容器管理 bean 的整个生命周期。通过反射实现对其他对象的控制，包括初始化、创建、销毁等，解放手动创建对象的过程，同时降低类之间的耦合度。

IOC 的好处：降低了类之间的耦合，对象创建和初始化交给 Spring 容器管理，在需要的时候只需向容器进行申请。

6.IOC 的优点是什么？

（1）IOC 和依赖注入降低了应用的代码量。

（2）松耦合。

（3）支持加载服务时的饿汉式初始化和懒加载。

7.什么是依赖注入？

在 Spring 创建对象的过程中，把对象依赖的属性注入到对象中。依赖注入主要有两种方式：构造器注入和属性注入。

8.Bean 的生命周期

（1）对 Bean 进行实例化

（2）给 bean 注入属性

（3）如果 Bean 实现了 aware 接口，会执行接口对应的方法。

（4）如果存在后置处理器，Spring 将调用它们的后置处理器的前置方法

（5）执行初始化方法

（6）如果存在后置处理器，调用后置处理器的后置方法（apo 就是在这一步进行的）

（7）Bean 初始化完成

9.BeanFactory 和 FactoryBean 的区别？

BeanFactory 是个 Factory，也就是 IOC 容器或对象工厂，FactoryBean 是个 Bean。在 Spring 中，所有的 Bean 都是由 BeanFactory(也就是 IOC

容器)来进行管理的。 但对 FactoryBean而言，这个 Bean 不是简单的 Bean，而是一个能生产或者修饰对象生成的工厂Bean, 它的实现与设计模式中的工厂模式和修饰器模式类似

10.Spring 自动装配的方式有哪些？

Spring 的自动装配有三种模式：byType(根据类型)，byName(根据名称)、constructor(根据构造函数)。

(1)byType

找到与依赖类型相同的 bean 注入到另外的 bean 中，这个过程需要借助 setter 注入来完成，因此必须存在 set 方法，否则注入失败。

(2)byName

将属性名与 bean 名称进行匹配，如果找到则注入依赖 bean。

11.@Autowired 和@Resource 的区别？

@Autowired 注解是按照类型（byType）装配依赖对象的,但是存在多个类型一致的 bean，无法通过 byType 注入时，就会再使用byName 来注入，如果还是无法判断注入哪个 bean 则会 UnsatisfiedDependencyException。 @Resource 会首先按照 byName 来装配，如果找不到 bean，会自动 byType 再找一次。

12.@Bean 和@Component 有什么区别？

都是使用注解定义 Bean。@Bean 是使用 Java 代码装配 Bean，@Component 是自动装配 Bean。

（1）@Component 注解用在类上，表明一个类会作为组件类，并告知 Spring 要为这个类创建 bean，每个类对应一个 Bean。

（2）@Bean 注解用在方法上，表示这个方法会返回一个 Bean。@Bean 需要在配置类中使用，即类上需要加上@Configuration 注解。

13.Spring 事务实现方式有哪些？

Spring 事务机制主要包括声明式事务和编程式事务。

使用 **@Transactional** 注解开启声明式事务。

15.有哪些事务传播行为？

常用的就 2 种，required 和 required new

16.required 和 required new 的区别:

（1）使用 required，内层事务与外层事务共用一个事务，出现异常同时回滚。

（2）使用 required new 时，内层事务与外层事务是两个独立的事务。一旦内层事务进行了提交后，外层事务不能对其进行回滚。两个事务互不影响。

17.Spring 的单例 Bean 是否有线程安全问题？

（1）有实例变量的 bean 是有状态的 bean，是非线程安全的。

（2）没有实例变量的对象是无状态的 bean，是线程安全的。

Bean 在多线程环境下不安全，一般用 Prototype 模式或者使用 ThreadLocal 解决线程安全问题。

MyBatis

1.#{}和${}的区别是什么？

#{} 是预编译处理，${}是字符串替换。

使用#{}可以有效的防止 SQL 注入，${}不可以

#{}传过来的参数带单引号''，而${}传过来的参数不带单引号。

2.mybatis 什么时候用的${}

动态表名字以及 ordery by 后面的字段，因为#{}传过来的参数带单引号''，而${}传过来的参数不带单引号。

3.当实体类中的属性名和表中的字段名不一样，怎么办？

（1）通过在查询的 sql 语句中定义字段名的别名，让字段名的别名和实体类的属性名一致。

（2）通过 resultmap 来映射字段名和实体类属性名的对应的关系。

4.Mybatis 是如何进行分页的？分页插件的原理是什么？

Mybatis 使用 RowBounds 对象进行逻辑分页，它是针对 ResultSet 结果集执行的内存分页。分页插件的基本原理是使用 Mybatis 提供的插件接口，实现自定义插件，在插件的拦截方法内拦截待执行的 sql，然后重写 sql，添加对应的物理分页语句和物理分页参数。

5.MyBatis 实现一对一有几种方式?具体怎么操作的？

有联合查询和嵌套查询。联合查询是几个表联合查询,只查询一次,通过在 resultMap 里面的 collection 节点配置一对多的类；嵌套查询是先查一个表,根据这个表里面的 结果的外键 id,去再另外一个表里面查询数据,也是通过配置 collection,但另外一个表的查询通过 select 节点配置。

6.MyBatis 实现一对多有几种方式,怎么操作的？

有联合查询和嵌套查询。联合查询是几个表联合查询,只查询一次,通过在 resultMap 里面的 collection 节点配置一对多的类就可以完成；嵌套查询是先查一个表,根据这个表里面的 结果的外键 id,去再另外一个表里面查询数据,也是通过配置 collection,但另外一个表的查询通过 select 节点配置

7.Mybatis 的一级、二级缓存

一级缓存: 基于 PerpetualCache 的 HashMap 本地缓存，其存储作用域为 Session，当 Session flush 或 close 之后，该 Session 中的所有 Cache 就将清空，默认打开一级缓存。

二级缓存与一级缓存其机制相同，默认也是采用 PerpetualCache，HashMap 存储，不同在于其存储作用域为 Mapper(Namespace)，并且可自定义存储源，默认不开启二级缓存缓存 。

8.Mybatis Mapper 接口支持重载吗？

不支持

因为 mybatis 是根据 package+Mapper+method 全限名作为 key，去 xml 内寻找唯一 sql 来执行的。如果方法名一样会导致找到两个 sql 报错。

SpringBoot

1.有什么好处

（1）简化了原来 spring 及 springmvc 一大堆配置文件

（2）嵌入的 Tomcat 不需要以 war 包形式部署

（3）进行 maven 依赖包版本统一管理

2. SpringBoot 自动配置的原理是什么？

主要是 Spring Boot 的启动类上的核心注解 SpringBootApplication 注解主配置类，有了这个主配置，类启动时就会为 SpringBoot 开启一个 @EnableAutoConfiguration 注解自动配置功能。

有了这个 EnableAutoConfiguration 的话就会：

从配置文件 META_INF/Spring.factories 加载可能用到的自动配置类

去重，并将 exclude 和 excludeName 属性携带的类排除过滤，将满足条件（@Conditional）的自动配置类返回。每个类又有自己的一个属性类，每个属性类的属性又从 springboot 的配置文件中读取。

JVM

1. 内存模型

堆 -Xms（最小堆内存）和-Xmx（最大堆内存）可以指定堆的大小,jdk1.8 把静态变量移到了堆中，字符串常量池 StringTable 也移到了堆中，堆又分为年轻代（复制算法清除垃圾，速度快，避免内存碎片，细分为伊甸园区和幸存者 0 区幸存者 1 区，年轻代内存比 8:1:1），和老年代（标记整理算法）这两块幸存者区域采用复制算法进行垃圾回收

虚拟机栈 每个方法被执行的时候都会创建一个栈帧

虚拟机栈由细分为局部变量表、操作栈、动态链接、方法方法放回地址

本地方法栈 虚拟机使用到的 Native（C 语言）方法服务

元空间 存储已被虚拟机加载的类的信息以及运行时常量池

程序计数器 当前线程所执行在那一行做一个计数

2. JVM 垃圾回收算法

（1）**标记-清除**：标记哪些要被回收的对象，然后统一回收但是会有两个主要问题：1.效率不高，标记和清除的效率都很低；2.会产生大量不连续的内存碎片。

（2）**复制算法**：速度快，没有内存碎片，复制算法将可用内存按容量划分为相等的两部分，然后每次只使用其中的一块，当一块内存用完时，就将还存活的对象复制到第二块内存上，然后一次性清除完第一块内存，再将第二块上的对象复制到第一块。缺点是要浪费内存，因为 8：1：1 有一半内存要浪费

（3）**标记-整理**：解决了内存碎片的问题，消除了复制算法当中，内存减半的问题，但是效率低于复制算法，并且会暂停用户线程

3. 垃圾回收器

需要学习两款垃圾回收器 G1 和 CMS，需要知道两者的区别，应用场景，以及两者是怎么工作的，这个需要自己去学，面试题写了没

学过也很难背。需要学习一款堆分析工具，mat，这两个都可以看尚硅谷宋红康 jvm 视频。

Springcloud

（了解即可，可以晚上学一下整理的视频，面试题常见就这些）

1.你知道哪些组件，都干嘛的

（1）Sentinel：流量控制、熔断降级、系统负载保护等多个维度保护服务的稳定性。

（2）Nacos：注册中心、配置管理。

（3）Openfign：服务之间的远程调用

（4）Seata：处理微服务分布式事务

2.Springboot 与 SpringCloud 区别

Spring boot 是 Spring 的一套快速配置脚手架，可以基于 Spring Boot 快速开发单个微服务；Spring Cloud 是一个基于 Spring Boot 实现的微服务开发框架。

Spring Boot 可以快速、方便集成单个个体,Spring Cloud 是关注全局的服务治理框架。

3.你知道有哪些注册中心

（1）ZooKeeper CP

（2）Eureka AP

（3）Consul CP

（4）Nacos CP+AP

4.什么是 cap 理论

一个分布式系统最多只能满足 CAP 中的两个条件，不可能同时满足三个条件

（1）C（Consistency）：这里指的是强一致性。保证在一定时间内，集群中的各个节点会达到较强的一致性，

（2）A (Avalibility)：可用性。意味着系统一直处于可用状态。个别节点的故障不会影响整个服务的运作

（3）P（Partition Tolerance)：分区容忍性。当系统出现网络分区等情况时，依然能对外提供服务。

Rocketmq，rabbitmq：记住几个常见面试题：消息丢失，重复消费，消息积压解决方法以及顺序消息怎么发送即可。