被面试官拷打八股，还是因为自己的八股准备不够充分，下阶段要着力于八股了！

1. java中有几个基本的数据类型?

byte、short、int、long、float、double、char、boolean(⚠️)

```
第一次没答出来boolean，忘记了，反思一下
```

2. 一个byte是几个字节?

1字节，8bit

3. 一个char类型数据可以转成什么类型?

int

追问：可以转换成byte吗?

不行(❌)

```
如果字符的Unicode码值在0到127之间，可以将char类型强制转换为byte类型)
```

4. 如何创建float类型的变量

数字后面加f

5. float与double类型进行混合运算结果会准确吗?

不会，混合运算时会将float转换为double后再进行运算，转换时会产生舍入误差，尤其是在运算小数部分时。

6. java中 double类型数据的组织形式是怎么样的?

有几位是表示数值部分，有几位表示它是2的几次方

```
  通过IEEE标准组织：具体来说：
  | 符号位（1 bit）| 指数位（11 bits）| 尾数位（52 bits）|
```

7. java中如何准确地表达数据的加减运算?

使用BigDecimal类

8. 你在你的项目中有用过这个数据类型吗?

没有用过(⚠️)

随便举个例子不就行了。。。
财务计算：BigDecimal可用于进行精确的货币计算，避免因浮点数运算引起的舍入误差。
　例如，在购物网站中，计算订单总金额、折扣、税费等涉及货币计算的地方都可以使用BigDecimal来进行精确计算。

9. java中一般用什么数据类型来接收 定义在数据库中的主键

Integer(❌)

踩大雷！！！！！！！！！！

在Java中，一般使用long或者Long数据类型来接收和定义数据库中的主键。

如果主键是自增长的整数类型，例如在MySQL中使用AUTO_INCREMENT属性，那么大部分情况下使用long类型是适合的。

如果主键可能为空或者允许为空，那么可以使用Long类型。Long是long的包装类，可以接受null值，而long类型无法直接接受null。

10. 数据库中的id用的什么数据类型?

int(⚠️)

⚠️一般常见的id数据类型选择：

整数类型（Integer Types）：在大多数情况下，整数类型是用来存储主键 id 的常见选择。例如，在MySQL中可以使用 INT、BIGINT、SERIAL 等数据类型来存储主键。在Oracle数据库中，可以使用 NUMBER 数据类型。

字符串类型（String Types）：有时候也会使用字符串类型来存储主键 id。例如，使用 VARCHAR 或 CHAR 数据类型。这种情况下，一般是由于主键的值并不是自增长的整数，而是具有特定的格式或需要满足一定的业务需求。

全局唯一标识符（GUID）或通用唯一标识符（UUID）：在某些情况下，可以使用全局唯一标识符或通用唯一标识符来作为主键 id。这些标识符是全局唯一的，并且可在不同系统之间保持唯一性。在数据库中可以使用 UNIQUEIDENTIFIER 类型来存储这样的主键。

11. 刚才说到Integer是一种包装类型，如果我定义一个Integer a = 1；在定义 int b = 1；我进行if(a == b)判断条件会成立吗?

会成立，因为1是有缓存的(⚠️虽然实际原理和我想得不一样，但是说出这句话是对的，属于运气好)

(实际上-128～127是自动拆箱(因为有缓存)，范围外的都是自动装箱后比较对象地址)

---这里，我以为会发生自动装箱，然而不是的，是拆箱。。。。。🥹，其实并不是，
正确的逻辑为：
在Java中，由于 Integer 是对象类型，而 int 是原始数据类型，对于数值在 -128 到 127 之间的整数，会在编译阶段自动进行装箱和拆箱，所以 if(a == b) 的判断条件会成立。

但是，对于大于 127 或小于 -128 的整数，由于 Java 中的自动装箱是缓存 Byte、Short、Integer、Long 以及 Character 这几种类型的对象，范围是 -128 到 127，所以 Integer a = 1000 和 int b = 1000 会创建两个不同的对象，即使它们的值相同。因此，if(a == b) 的比较条件不会成立。

为了比较 Integer 对象和 int 值的相等性，建议使用 .equals() 方法进行比较，如 if(a.equals(b)) 或 if(b == a.intValue())。这样可以确保正确比较值的相等性而非引用的相等性。

12. 加入 Integer a = new Integer(1); int b = 1; 那么if(a == b)是否相等?

不会相等，因为a这边在堆中创建了一个新的对象

而b则在自动装箱的时候使用了缓存的Integer(1)，因此不相等

13. 刚才说的等号两边一边是Integer，一边是int，为什么能这样判断，你了解吗?

因为java有自动装箱、自动拆箱的功能，像刚才的情况会自动把int包装成Integer进行比较!

⚠️注意看11.的解释，一定要将其理解!

14. java中 boolean类型可以转换成int类型吗?

不行(❌)

寄！刚知道!
🍡当将 boolean 类型的值转换为 int 类型时，true 转换为 1，false 转换为 0。这是因为 boolean 类型只有两个可能的取值：true 和 false。
boolean 类型转换为 int 类型是隐式的，可以直接赋值或使用条件运算符进行转换。在此过程中，Java会自动将其转换为对应的整数值。

15. java集合类中，线性方式存储的有哪些?

ArrayList、LinkedList (⚠️)

还有
Vector：Vector 是传统的动态数组，与 ArrayList 类似，但是被设计为线程安全的，它支持同步操作。
Stack：Stack 是堆栈数据结构的 Java 实现，它基于动态数组。它使用后进先出 (LIFO) 的方式存储和访问元素。

追问：ArrayList、LinkedList底层的实现是什么?

ArrayList底层是数组，LinkedList底层是链表

追问：(ArrayList)数组的大小是怎么确定的呢?

在构造函数中，如果没给值就按默认大小创建，如果给了并且合法就按提供的值来创建

追问：假设ArrayList容量为5，add到第几个数据的时候扩容?

满了就扩容(❌)

初始容量为5的 ArrayList 可以容纳5个元素，它会一直使用这个固定的容量直到添加第6个元素。

追问：ArrayList、LinkedList的使用场景是什么?

ArrayList频繁查询、LinkedList频繁增删

如果需要频繁机访问或在末尾进行添加/删除操作，可以使用 ArrayList；如果需要频繁在中间位置进行插入/删除操作或实现栈/队列相关功能，可以使用 LinkedList

追问：LinkedList底层如何删除一个数据?

就是链表的节点删除

16. 讲一下平衡二叉树

1. 是一种特殊的二叉搜索树
2. 每个节点的左子树和右子树的高度差不超过1
插入、删除节点时，会通过左旋、右旋等操作维持平衡

17. 了解红黑树吗

红黑树是一种自平衡的二叉搜索树，类似于平衡二叉树

18. HashMap的实现方式是什么样的?

1.首先，计算键的哈希值。哈希函数（Hash Function）负责将键转换成一个整数值，一般情况下会根据键的特征进行计算，目的是使哈希值尽可能均匀地分布在桶的范围内。
2.根据哈希值计算出桶的索引位置。通常使用一个取模运算将哈希值映射到桶的范围内，得到桶的索引位置。
3.在桶的索引位置上进行操作。如果桶为空，表示没有冲突，直接将键值对存储在该位置上；如果桶不为空，表示发生了冲突，可能有多个键值对哈希值映射到同一个桶上。

19. 从数据结构角度，除了链表法还有什么解决Hash冲突的方式?

(❌)，这里搞错题意了

1.开放定址法（Open Addressing）：在桶的索引位置发生冲突时，该方法通过探测其他空桶来解决冲突。具体有以下几种探测方式：
  线性探测（Linear Probing）：在发生冲突的位置后依次检查下一个位置，直到找到空桶。
  二次探测（Quadratic Probing）：根据一个探测序列公式，依次探测下一个位置，直到找到空桶。
  双重哈希（Double Hashing）：使用第二个哈希函数来计算下一个位置的偏移量，直到找到空桶。
开放定址法的优点是可以充分利用桶的空间，没有额外的链表开销，但它也容易引发聚簇（Clustering）现象，导致性能下降。

2.再哈希法（Rehashing）：在桶的索引位置发生冲突时，该方法通过再次应用哈希函数来计算新的索引位置，直到找到空桶为止。通常会使用一个不同的哈希函数，以减少冲突的可能性。
再哈希法的优点是可以在不同的哈希函数之间切换，以克服某个哈希函数的缺点。但它也需要维护多个哈希函数，并且可能增加插入和查找操作的时间。

🌟总结来说，再哈希法是在发生冲突时使用不同的哈希函数重新计算索引位置，而二重哈希是使用第二个哈希函数计算下一个位置的偏移量。

20. 说几个你熟悉的设计模式

单例模式、工厂模式、代理模式

追问：如何写一个单例模式?

追问：多线程创建单例，会不会不问题，怎么解决?

(❌)这里我没有说清楚，就记得两次判断null了。。

用双重检查法：
```
public class Singleton {
    private volatile static Singleton instance;

    private Singleton() {}

    public static Singleton getInstance() {
        if (instance == null) {
          🌟synchronized (Singleton.class) {
                if (instance == null) {
```

```
                    instance = new Singleton();
                }
            }
        }
        return instance;
    }
}
```

21. 说一下volatile关键字的作用

只说了1、2(⚠️)

1、可见性（Visibility）：在多线程环境下，对于被 volatile 关键字修饰的变量，当一个线程修改了该变量的值，其他线程能够立即看到这个变化。这是因为通过 volatile 关键字修饰的变量会直接在主内存中进行读取和写入，而不会使用线程的本地缓存。因此，volatile 变量可以保证线程之间的变量值一致性。

2、禁止指令重排序（Prevent Instruction Reordering）：在编译器和处理器的优化过程中，会对指令进行重排序，以获得更好的性能。然而，在某些情况下，指令重排序可能会导致多线程程序的运行结果出现错误。通过将变量声明为 volatile，可以禁止编译器和处理器在对该变量操作的时候进行重排序，保证指令的顺序性。

3、部分原子性（Partial Atomicity）：虽然 volatile 关键字不能保证所有操作的原子性，但是对于简单的读取和写入操作是原子的。也就是说，对于 volatile 变量的读取和写入，可以保证其操作具有原子性。但是对于复合操作，如 volatileCount++ 这样的自增操作，并不能保证原子性，需要额外的同步手段，如使用 synchronized 关键字或使用原子类来实现。

22. 你对synchronize的理解是怎么样的呢?

可以给代码块或者方法上锁

追问：可以对哪些结构加锁?

方法和代码块

追问：synchronize是怎么锁住一个对象的?

通过获取对象的锁

Java 中的对象锁是通过对象头中的一些字段来实现的。每个对象在内存中都有一个对象头，在 HotSpot 虚拟机中，对象头包含了两部分内容：Mark Word 和 Class Metadata Address。其中，Mark Word 中的一些位用于存储锁的相关信息。
在 HotSpot 虚拟机中，对象锁有两种状态：无锁状态和重量级锁状态。
对于无锁状态的对象，Mark Word 中的锁信息部分是空的，可以被任意线程访问，不存在互斥。
当第一个线程访问一个无锁状态的对象时，它会尝试使用 CAS（Compare And Swap）操作来将对象的 Mark Word 修改为自己的线程 ID，同时将锁标志位设置为 1，表示获取锁成功。
如果 CAS 操作失败，说明有其他线程已经获取了锁，此时当前线程会进入自旋（Spin）等待，不断尝试 CAS 操作获取锁，而不是让线程进入阻塞状态。
当自旋次数达到一定阈值，或者其他线程释放了锁，当前线程成功获取到了锁，它将会将锁的标志位置为 0，表示无锁状态。
当多个线程竞争同一个对象的锁时，如果尝试获取锁的线程较少，自旋等待的方式可以减少线程切换的开销，提高运行效率。但如果竞争线程较多，自旋等待会占用大量的 CPU 时间，造成资源浪费。
当一个线程多次自旋仍然无法获取锁时，系统会自动升级为重量级锁，即使用悲观锁的方式进行实现。
重量级锁是一种在内核层面实现的锁，在锁的竞争激烈时，没有获取到锁的线程会进入阻塞状态，让出 CPU 给其他线程使用。当锁释放时，被阻塞的线程会被唤醒，重新竞争锁。
总结来说，Java 中的对象锁是通过对象头中的字段来实现的，通过 CAS 操作和自旋等待实现轻量级锁，当竞争激烈时升级为重量级锁，通过阻塞线程实现同步。不同的锁状态带来的开销和性能表现也不同，需要根据具体的场景选择合适的锁机制。

23. 了解偏向锁吗?

不了解(❌，真看过，但真忘了)

偏向锁（Biased Locking）是 Java 中锁优化的一种技术，旨在减少无竞争情况下的锁操作的开销。
在没有线程竞争的情况下，偏向锁能够帮助提高程序的性能。
　它的核心思想是：如果一个线程获得了对象的锁，并且在后续的执行过程中没有其他线程来竞争锁资源，那么该线程可以一直保持对该对象的偏向。
　这样，下次该线程再次请求锁资源时，就不需要再进行同步操作，直接获取即可。这样就避免了无竞争情况下的不必要的锁操作，提升了程序的性能。

*偏向锁的实现机制是：
　在对象头中的 Mark Word 中的某个标志位记录了线程 ID，表示当前对象的锁已经被偏向于该线程。当第一个线程访问一个对象时，它会尝试将对象头的 Mark Word 修改为自己的线程 ID，并将偏向锁标识位设置为 1，表示获取锁成功。
如果 CAS 操作成功，那么该线程就获得了对象的偏向锁，并且进入偏向状态。之后，如果该线程再次请求锁资源，只需要检查一下 Mark Word 的线程 ID 是否与自己相同即可，非常高效。这个过程不需要释放锁资源。
但如果有其他线程竞争同一个对象的锁，偏向锁就会失效。当出现竞争情况时，偏向锁会自动升级为轻量级锁或重量级锁，取决于竞争的情况。
总结来说，偏向锁是一种针对无竞争情况下的锁优化技术，能够减少不必要的锁操作，提高程序的性能。它通过在对象头中记录线程 ID，避免了重复获取锁的同步操作。但在有竞争的情况下，偏向锁会自动失效，升级为其他类型的锁来保证线程安全。
👍看完后就明白了，短期内绝对忘不了！

24. 解释一下JVM内存模型(分区情况，然后再讲一下垃圾回收情况)

- 程序计数器
- Java 虚拟机栈
- 本地方法栈
- Java 堆
- 元数据区

🌟**追问**：我们创建一个对象，可能会在哪些区域分配内存?

堆

🌟**追问**：一定吗?

(❌)这块确实原本不知道，乱说一通

追问：你有了解过栈上分配吗？(这不是明显的提示吗？奈何我真不会)

> 🍡栈上分配（Stack Allocation）。栈上分配是一种优化技术，它将对象分配到栈内存而不是堆内存中，以提高程序的性能。
>
> 在传统的 Java 内存模型中，对象的分配发生在堆内存中，需要进行动态内存分配和垃圾回收。然而，栈上分配通过将对象分配到栈帧中，即方法调用栈，来减少内存管理的开销。
>
> 栈上分配适用于满足以下条件的对象：
> ✦对象是线程私有的，即不会被其他线程访问或共享。
> ✦对象的生命周期在方法内部，即对象的引用不会逃逸出方法（不会被方法外部的代码所引用）。
> ✦对象是值对象（Value Object），即对象在存储和使用上与原生数据类型类似，没有复杂的生命周期或引用关系。
>
> 栈上分配的好处包括：
> ✦分配和回收对象的开销更低：栈上分配不需要进行堆内存的动态分配和垃圾回收，因此可以减少运行时的开销。
> ✦对象可以更快地释放：当方法调用结束时，栈帧中的对象会随着栈帧的弹出而自动释放，无需等待垃圾回收。
>
> 需要注意的是，栈上分配对于大部分对象来说并不适用，因为大部分对象的生命周期比一个方法调用更长，并且可能被多个线程访问。栈上分配通常用于创建小型、临时的对象，例如局部变量或方法参数。
>
> 总而言之，栈上分配是一种优化技术，在满足特定条件下，将对象分配到栈内存中，以减少内存管理的开销和提高程序性能。

25. 堆中有新生代和老年代，什么样的对象会在新生代，什么样的对象会在老年代呢?

我回答的原话是：频繁创建销毁的对象会在新生代中(⚠️这样表达非常不准确，下次思考2秒再张嘴说话！！)

新创建的对象在新生代，新生代中年龄到达设定值或者放不下的情况下会放到老年代中

26. 哪些对象能够被回收，回收的机制是什么?

首先找到GC Roots，再找他们与他们有直接引用的对象，这些是不会被回收的

可达性分析

🌟**追问**：你刚提到了GC Root， 哪些对象能够作为GC Root?

(❌，真看过！也真忘了！😡)

```
在 Java 中，以下几种对象可以作为 GC Roots：

✦虚拟机栈（栈帧中的本地变量表）中引用的对象；
✦方法区中类静态属性引用的对象；
✦方法区中常量引用的对象；
✦被 JNI（Java Native Interface，即 Java 本地接口）引用的对象；
✦线程中正在执行的方法的局部变量或输入参数引用的对象。
```

27. mysql索引是什么结构?

B+树

追问：描述一下B+树

叶子节点存放数据，非叶子节点存放引用，底层叶子结点间用双向链表连接.....

追问：底层双向链表的作用是什么?

方便顺序访问，查询一片区域是效率比较高，减少了随机I/O(⚠️感觉说得不是很好，看看下面的补充吧)

```
✦顺序访问：双向链表使得数据页可以按照索引顺序进行顺序访问。这对于一些范围查询或者顺序扫描操作很重要，因为它可以减少磁盘 I/O 的次数，提高查询性能。
✦快速查找：在双向链表中，每个数据页都有指向前一个和后一个数据页的指针。这样可以通过两个方向进行查询，即根据索引的顺序可以迅速找到下一个数据页或者前一个数据页。这对于快速定位到指定数据页并获取相应的数据非常重要，减少了查询的时间复杂度。
✦索引维护：当进行数据插入或删除操作时，双向链表可以快速定位到叶子节点，并调整链表指针来保持索引的有序性。例如，插入操作会将新的数据插入到正确的位置，并更新相邻叶子节点的链表指针。这样可以保持索引的有效性，并减少维护索引的开销。
```

28. 讲一下聚簇索引

```
在聚簇索引中，数据行按照索引的顺序存储在磁盘上。
以下是关于聚簇索引的一些重要特点和优势：
✦数据和索引的结合：聚簇索引将索引行和实际数据行映射到同一块存储空间上。这样，在查询时可以通过索引快速定位到符合条件的数据行，减少了磁盘 I/O 操作的次数，提高了查询的效率。
✦数据的物理有序性：由于聚簇索引定义了物理数据存储的顺序，因此相邻的数据行通常在磁盘上也是相邻存储的。这种有序性可以提高查询范围和顺序访问的性能，因为可以减少磁盘 I/O 操作和随机访问的次数。
✦压缩优势：聚簇索引可以获得更好的数据压缩效果。由于相邻的数据行通常具有相似的值，所以利用聚簇索引可以实现更高的数据压缩率。这对于占用大量存储空间的大表尤为重要，可以减少存储和 I/O 开销，并提高整体数据库性能。
```

29. 表里有主键id列，name列，我建立(id, name)的联合索引会起作用吗？有必要吗?

30. mysql事务有几种隔离级别，你了解过吗?

(❌，非常低级的失误，这个问题我答的很差，就是没背下来，下次不能再背不出来了)

1.读未提交（Read Uncommitted）：最低的隔离级别，它允许一个事务读取另一个事务未提交的数据。这种隔离级别可能会导致脏读（Dirty Read），即读取到未提交的数据。
2.读已提交（Read Committed）：在该隔离级别下，一个事务只能读取到已经提交的数据。这可以避免脏读情况的发生，但可能会导致不可重复读（Non-repeatable Read），即在同一个事务内两次读取同一个数据得到不同的结果。
3.可重复读（Repeatable Read）：在该隔离级别下，保证了同一个事务内多次读取同一数据的结果是一致的。其他事务对该数据的修改只能在当前事务提交后才能看到。这可以避免脏读和不可重复读，但可能会导致幻读（Phantom Read），即在同一个事务内两次查询得到不同的结果。
4.序列化（Serializable）：最高的隔离级别，它通过强制事务串行执行来避免脏读、不可重复读和幻读的情况。在此隔离级别下，事务之间的并发性大大降低，可能会对系统的性能产生较大的影响。

31. innodb怎么实现可重复读?

通过快照实现(MVCC都忘了说了，获取五个事务相关的状态码也忘说了。。。获取最小事务id，当前活跃事务id列表、创建下一个的事务会使用到的id、、、)

生成读视图：在每个事务启动时，InnoDB 会为该事务生成一个唯一的读视图（Read View）。读视图中包含了该事务在开始时所有活跃的事务ID（有可能会修改到的数据行）。读视图决定了事务在执行期间看到的数据版本。
读取快照数据：在事务的执行期间，InnoDB 通过对比读视图和数据行的版本信息来确定哪个数据版本是可见的。对于未提交的事务，InnoDB 不会将其修改的数据行对当前事务可见。相反，InnoDB 使用数据行的旧版本来提供一个可重复读的视图。
锁定被修改的数据：为了保证每个事务的修改不会被其他事务读取，InnoDB 在可重复读隔离级别下会为 SELECT 查询语句下的读操作（如 SELECT...FOR UPDATE）设置锁。这样，其他事务无法修改或读取被锁定的数据，从而保证了可重复读。

32. 加入给你一个sql语句，你会从哪几个方面考虑优化呢?

(❌)我直接说用explain查看索引使用情况，，，然后explain里的参数还不熟悉，，，挺打脸的，抓紧背!

✧查看索引是否合理：考虑查询语句中使用的字段是否都有合适的索引。缺乏索引或者索引不合理可能导致全表扫描或者临时表的创建，从而导致性能下降。

✧优化查询条件：检查查询语句中的 WHERE 条件是否能够有效地过滤掉不符合条件的数据，从而减少扫描的数据量。可以考虑对查询条件进行索引覆盖，使用合适的索引来加快查询。

✧避免使用通配符：%和_ 通配符会导致索引的失效，可以考虑使用前缀索引或者全文索引来代替。

✧优化 JOIN 操作：如果查询语句中包含 JOIN 操作，需要确保关联的字段上都有索引，并且 JOIN 条件尽可能简单。可以考虑使用合适的 JOIN 类型，例如 INNER JOIN、LEFT JOIN 或者使用 EXISTS 子查询来优化。

✧避免使用子查询：子查询可能会导致性能问题，可以考虑使用 JOIN 或者其他方式来重写查询语句，减少子查询的使用。

✧减少数据传输量：只选择所需的字段，避免返回不必要的数据。可以使用 SELECT 子句的列名列表，而不是使用通配符。

✧分页查询优化：对于分页查询，可以使用 LIMIT 和 OFFSET 进行控制。对于大数据量的分页查询，可以考虑使用游标（cursor）方式，避免一次性加载所有数据。

✧配置合理的缓存和缓冲区：根据具体情况，调整 MySQL 的缓存大小和缓冲区的配置，以提升查询性能。

✧定期进行表的统计和优化：使用 ANALYZE TABLE 或者 OPTIMIZE TABLE 命令，以及常规的表维护工作，保持表的性能。

注意，具体的优化策略需要根据具体的情况来确定，可以通过查看执行计划、使用 Explain 命令、分析慢查询日志等手段来定位性能瓶颈，然后针对性地进行优化。

33. 在explian报告中，有哪些常见的索引使用方式？举几个例子

(❌)没背，说的中文，巨尬

type 列：表示查询访问方法，常见的类型有 const、eq_ref、ref、range、index、all 等。其中，const 表示常量查询，eq_ref 表示使用唯一索引查询，ref 表示非唯一索引查询，range 表示范围查询，index 表示索引覆盖查询，all 表示全表扫描。

34. 索引覆盖是什么？

通过索引覆盖，查询可以在索引中直接定位到所需的数据，避免了额外的磁盘IO和内存操作，因此可以显著降低查询的消耗。

35. where条件是如何过滤掉数据的？在哪一层？

(❌)这个真不太懂，需要事后了解一下

Storage Engine 层：MySQL的存储引擎 （如InnoDB、MyISAM等）负责管理数据文件的存储和访问。当查询执行时，存储引擎首先根据WHERE条件的索引信息，在存储引擎层利用索引来提供符合WHERE条件的数据行。这一步可以通过索引的B+树结构快速定位符合WHERE条件的数据行。
MySQL Server 层：在存储引擎层返回数据行给MySQL Server之后，MySQL Server对返回的数据执行额外的过滤操作。这些过滤操作包括JOIN、GROUP BY、HAVING等操作。MySQL Server会对返回的数据进行进一步处理，以根据查询的其它条件进行筛选和聚合。
在这一层，MySQL将会执行更复杂的查询操作，如连接多个表数据、聚合函数计算等。WHERE条件过滤操作的结果将作为MySQL Server中进一步的操作的输入。
所以，在MySQL中，WHERE条件在存储引擎层和MySQL Server层都起到了关键的作用。存储引擎通过索引快速定位并返回符合条件的数据行，然后MySQL Server对返回的数据进行进一步的处理和过滤，以满足查询的需求。这样的设计可以更好地利用索引和减少数据的传输量，提高了查询的效率。

36. A列 建有索引 B列没有 查询where A = 1 and B = 1时底层发生了什么?
37. MySQL 会首先检查 A 列的索引，通过索引的 B+树结构迅速定位到满足条件 A = 1 的数据行。
38. 对于满足条件 A = 1 的数据行，MySQL 将进一步检查每一行的 B 列的值是否等于 1。由于 B 列没有索引，MySQL 需要对A=1的数据进行全表扫描来验证 B 列的值是否满足条件 B = 1。

37. redis中有几种数据类型？你在你的项目中用过哪几种数据结构?

(❌，没背，不会)

分布式锁就用到了String类型

常见的：⬇️
字符串 (String)：存储一个字符串值。可以存储任何类型的数据，包括数字、文本、二进制数据等。

列表 (List)：按照插入顺序存储一个有序的元素集合。可以在列表的两端进行插入和删除操作。

集合 (Set)：存储一个无序的、不重复的元素集合。可以对集合进行交集、并集、差集等常见集合操作。

有序集合 (Sorted Set)：类似于集合，但每个元素都关联了一个分数，被用来按照分数的顺序存储和获取元素。可以根据分数范围或成员来获取元素。

哈希表 (Hash)：存储键值对的无序散列表。适用于存储对象的相关字段，可以针对单个字段进行读取和更新操作。

38. redis 分布式锁是怎么实现的？原理是什么?

(⚠️)

分布式锁的实现原理如下：
✦使用 SETNX (set if not exist) 命令来尝试将一个唯一的标识（如锁的名字）作为 key，获取锁的资源。
✦当 key 不存在时， SETNX 命令会将该 key 设置为指定的值，并返回 1，表示获取锁成功。
✦当 key 已经存在时， SETNX 命令不会对 key 进行任何操作，直接返回 0，表示获取锁失败。
✦在获取锁成功后，为了避免锁永久占用，可以设置一个合适的过期时间 (expire)。可以使用 EXPIRE 命令或者 SET 命令配合 NX（仅在 key 不存在时设置）选项来设定过期时间。
✦在对锁进行解锁操作时，需要使用 DEL (delete) 命令来删除该锁的 key。

分布式锁的特点和使用注意事项：
✦分布式锁是通过 Redis 单线程的特点来保证原子性操作的，保证同一时刻只有一个客户端能够获取到锁。
✦可以使用 SETNX+EXPIRE 命令的组合来实现基本的分布式锁。
✦获取到锁后，执行具体的业务逻辑，然后再进行解锁操作，避免锁被长时间占用。
✦避免获取锁的客户端执行时间过长，导致锁过期后其他客户端获取到锁执行相同的操作。可以根据业务情况合理调整锁的过期时间。

39. 描述一下TCP建立连接的过程。

TCP（Transmission Control Protocol）是一种面向连接的可靠传输协议。下面是 TCP 建立连接的过程：

客户端发送 SYN（同步）报文：当客户端要与服务器建立 TCP 连接时，它会发送一个 SYN 报文给服务器。这个报文包含一个序列号（Seq）用于标识发送的数据段。

服务器回复 SYN-ACK（同步-确认）报文：服务器接收到客户端的 SYN 报文后，会向客户端返回一个 SYN-ACK 报文。这个报文中，服务器的序列号被设置为服务器初始序列号，确认号（Ack）被设置为客户端的序列号加 1，表示服务器已经收到了客户端的 SYN 报文。

客户端发送 ACK（确认）报文：客户端接收到服务器的 SYN-ACK 报文后，会向服务器发送一个 ACK 报文。该报文的序列号被设置为客户端的初始序列号，确认号被设置为服务器的序列号加 1，表示客户端已经收到了服务器的 SYN-ACK 报文。

连接建立：服务器接收到客户端的 ACK 报文后，双方成功建立了 TCP 连接。双方现在可以开始交换数据。

这个过程通常被称为三次握手，因为它包含了三个主要步骤，分别是客户端发送 SYN、服务器回复 SYN-ACK、客户端发送 ACK。三次握手的目的是确保双方都能够收发数据，并同步初始化序列号。

在 TCP 连接的过程中，每个报文都有对方的确认号和序列号，用于保证可靠性和顺序性。三次握手过程中，服务器和客户端都会分配初始序列号，以确保连接的唯一性和安全性。

40. 在一台主机的一个端口同时建立TCP和UDP协议，能建立起来吗?

(❌，计网还没看，说得七七八八的，太差了)

TCP（Transmission Control Protocol）是一种面向连接的可靠传输协议。下面是 TCP 建立连接的过程：

客户端发送 SYN（同步）报文：当客户端要与服务器建立 TCP 连接时，它会发送一个 SYN 报文给服务器。这个报文包含一个序列号（Seq）用于标识发送的数据段。

服务器回复 SYN-ACK（同步-确认）报文：服务器接收到客户端的 SYN 报文后，会向客户端返回一个 SYN-ACK 报文。这个报文中，服务器的序列号被设置为服务器初始序列号，确认号（Ack）被设置为客户端的序列号加 1，表示服务器已经收到了客户端的 SYN 报文。

客户端发送 ACK（确认）报文：客户端接收到服务器的 SYN-ACK 报文后，会向服务器发送一个 ACK 报文。该报文的序列号被设置为客户端的初始序列号，确认号被设置为服务器的序列号加 1，表示客户端已经收到了服务器的 SYN-ACK 报文。

连接建立：服务器接收到客户端的 ACK 报文后，双方成功建立了 TCP 连接。双方现在可以开始交换数据。

这个过程通常被称为三次握手，因为它包含了三个主要步骤，分别是客户端发送 SYN、服务器回复 SYN-ACK、客户端发送 ACK。三次握手的目的是确保双方都能够收发数据，并同步初始化序列号。

在 TCP 连接的过程中，每个报文都有对方的确认号和序列号，用于保证可靠性和顺序性。三次握手过程中，服务器和客户端都会分配初始序列号，以确保连接的唯一性和安全性。

41. HTTP协议的格式能描述一下吗？比如请求和响应头的格式。

(❌)说得七七八八

HTTP 请求的格式：
✦请求行（Request Line）：
    <请求方法> <目标资源路径> <协议版本>
✦请求头（Request Headers）：包含了关于请求的一些附加信息，格式为键值对，每个键值对占据一行。
    <键1>: <值1>
    <键2>: <值2>
    ...
✦空行（空格 + 回车换行）：
    用于分隔请求头和请求体。
✦请求体（Request Body）：
    可选的，适用于一些特定的请求，如 POST 请求，用于传输用户提交的数据。

HTTP 响应的格式：
✦状态行（Status Line）：
    <协议版本> <状态码> <状态信息>
✦响应头（Response Headers）：
    包含了关于响应的一些附加信息，格式为键值对，每个键值对占据一行。
    <键1>: <值1>
    <键2>: <值2>
    ...
✦空行（空格 + 回车换行）：
    用于分隔响应头和响应体。

✦响应体（Response Body）：
包含了服务器响应的数据。

42. GET和POST两种请求的区别是什么?

只说了1(⚠️)

1.参数传递位置：GET请求将参数附加在URL的末尾，形成URL参数，即在请求行中传递参数；而POST请求将参数包含在请求的消息体中，通过请求体传递参数。

2.数据传输方式：GET请求以明文形式传输参数，将参数添加在URL中，可以在浏览器的地址栏直接看到；POST请求则使用消息体传输参数，对参数进行了封装，不会直接暴露在URL中，相对较为安全。

3.请求长度限制：GET请求对URL的长度有限制，一般在浏览器中限制为2048个字符；而POST请求没有严格的长度限制，可以传输更大的数据量。

4.缓存机制：GET请求可以被缓存，而POST请求默认不会被缓存。

5.请求语义：GET请求主要用于获取资源，操作不应带来副作用，例如查询数据；POST请求主要用于向服务器提交数据，可能会对服务器产生影响，例如提交表单或创建新资源。

43. 说几个常见的HTTP相应码

(⚠️)只答了几个，并且不太熟悉

200 OK：表示请求成功，服务器成功处理了请求。

301 Moved Permanently：永久重定向，表示请求的资源已经被移动到了新的URL。

302 Found：临时重定向，表示请求的资源暂时被移动到了新的URL。

400 Bad Request：表示客户端的请求语法错误或无效，服务器无法理解。

401 Unauthorized：表示请求需要身份验证，客户端需要提供有效的凭据。

403 Forbidden：表示服务器理解请求，但拒绝执行，客户端没有访问权限。

404 Not Found：表示服务器无法找到请求的资源。

500 Internal Server Error：表示服务器内部错误，无法完成请求。

502 Bad Gateway是指作为代理或网关的服务器从上游服务器接收到一个无效的响应，导致无法完成请求。

503 Service Unavailable：表示服务器暂时无法处理请求，通常是由于服务器过载或维护。

401 vs 403

总结起来，401 Unauthorized表示客户端未提供有效的身份验证信息，需要进行身份验证；而403 Forbidden表示客户端已提供有效的身份验证，但是被服务器拒绝访问该资源。

44. 比如你遇到了502错误，你该怎么去排查错误呢?

首先因为5开头，确定是服务器方面的错误，然后去服务器查看报错信息以及日志信息。

(⚠️)

了解错误码：首先阅读错误信息，确切了解错误的类型和含义。不同的错误码代表不同的问题，了解错误码可以帮助你更快定位问题。

检查网络连接：确保你的网络连接正常。尝试访问其他网站或服务，以确认是否只有特定的网站或服务受到影响。

刷新页面或重试：有时候错误可能只是暂时的，尝试刷新页面或重新执行操作，看看错误是否会消失。

检查URL和请求参数：确保URL和请求参数正确无误。可能是由于请求的URL或参数不正确导致服务器返回错误。

检查服务器状态：如果错误是与特定的网站或服务相关，可以尝试访问其他网站或服务来确认是服务器端的问题还是客户端的问题。

查看服务器日志：如果你有服务器的访问权限，可以查看服务器的日志来获取更多的错误信息。服务器日志通常可以提供有关错误发生的具体细节，帮助你定位问题。

联系网站管理员或技术支持：如果你已经尝试过以上的排查步骤但问题仍然存在，建议联系网站的管理员或技术支持人员。他们可能具有更深入的技术知识和工具来帮助你解决问题。

45. 进程和线程的区别是什么?

定义：进程是一个正在执行的程序实例，具有独立的内存空间和资源。线程是进程的一部分，用于执行进程内的具体任务。

资源占用：每个进程都有自己独立的内存空间和资源，包括文件、网络线程、内核数据结构等。线程共享进程的资源，包括内存空间、文件描述符等。

调度和执行：进程在操作系统中被调度为可执行状态，并行或交替执行。线程在进程内被调度执行，共享进程的执行环境。

交互和通信：进程之间通常通过进程间通信（IPC）进行交互和通信，如管道、消息队列、共享内存等。线程之间由于共享相同的内存空间，可以直接访问和修改共享变量，实现线程间的通信和同步。

异常和崩溃：进程之间具有强隔离性，一个进程的崩溃不会影响其他进程。线程共享同一个进程的资源，一个线程的崩溃可能会影响整个进程的稳定性。

创建和销毁：创建和销毁进程需要操作系统的调用，涉及加载和卸载程序、分配和释放内存等操作。线程的创建和销毁更轻量级，通常由程序代码直接调用线程库函数进行操作。

46. OS中有一个轻量级进程的概念，你有了解吗?

(❌，不了解，赶紧了解一下)

轻量级进程是一种在操作系统中实现的并发执行的方式。它是在用户空间层面上模拟了传统操作系统中的进程，具有自己的执行上下文、栈、寄存器集等，但在内核层面上实际上共享了同一个真实的操作系统线程。

LWP的主要优势在于它比传统的进程更轻量级，创建和销毁一个LWP的成本要低于创建和销毁一个完整的进程。由于LWP共享了操作系统线程，它们之间的切换成本也更低。这使得创建和切换LWP的开销较小，且在并发执行和并行执行方面具有更高的效率。

一些操作系统或库会使用轻量级进程来实现并发，例如在Solaris操作系统中使用线程和LWP的模型来实现并发。此外，跨平台的库，如Java的线程模型，也使用了轻量级进程的概念。

需要注意的是，轻量级进程是在用户空间中实现的概念，而非操作系统内核的一部分。操作系统仍然负责线程和进程的管理，而轻量级进程是在用户空间层面上实现并发的机制。

47. 解释一下零拷贝是什么。

(❌，不会)

零拷贝（Zero Copy）是一种优化技术，用于在数据传输过程中减少不必要的数据拷贝操作，从而提高性能和效率。

传统的数据传输方式中，数据通常需要在不同的缓冲区之间进行多次拷贝操作。例如，在从磁盘读取数据并发送到网络时，数据需要从磁盘读取到内核缓冲区，然后再从内核缓冲区拷贝到用户空间缓冲区，最后才能通过网络发送出去。这样的拷贝操作会占用CPU时间和内存带宽，降低系统性能。

而零拷贝技术通过减少或消除不必要的数据拷贝，提高了数据传输的效率。它通过直接在内核空间中进行数据传输，避免了在用户空间和内核空间之间拷贝数据的过程。

具体来说，零拷贝的实现可以通过以下方式之一：

DMA（Direct Memory Access）技术：使用DMA控制器将数据直接从存储设备传输到网络适配器，或者从网络适配器直接传输到存储设备，避免了数据在内存中的中间拷贝。

内核缓冲区直接传输：在系统调用期间，将数据从内核缓冲区传输到网络适配器，或者从网络适配器传输到内核缓冲区，避免了在用户空间和内核空间之间的数据拷贝。

内核空间到用户空间的零拷贝：通过使用mmap（内存映射）或其他技术，将内核空间的数据直接映射到用户空间，而不需要数据在内存中的额外拷贝。

零拷贝技术可以在高速数据传输、存储系统和网络应用等场景中发挥作用。它能够减少数据拷贝带来的CPU负载和内存带宽的消耗，提高数据传输的效率和性能。

48. I/O多路复用你了解多少

(❌，乱讲一通 = 不了解)

I/O多路复用是一种高效的I/O处理机制，它允许单个线程或进程同时监听多个I/O事件，从而实现对多个I/O操作的并发处理。在I/O多路复用模式下，可以在一个线程中同时处理多个客户端请求而无需创建额外的线程。

追问：有几种I/O模式，有什么区别?

同步异步

阻塞式I/O（Blocking I/O）：在阻塞式I/O中，当应用程序发起一个I/O操作后，它会一直阻塞等待直到操作完成。这意味着应用程序无法继续执行其他任务，直到I/O操作完成。阻塞式I/O适用于简单的应用场景，但在面对并发处理需求或高负载情况下，可能会导致性能瓶颈。

非阻塞式I/O（Non-Blocking I/O）：在非阻塞式I/O中，当应用程序发起一个I/O操作后，它可以立即返回而不需等待操作完成。应用程序可以继续执行其他任务，然后通过轮询或其他方式来检查I/O操作是否完成。非阻塞式I/O可以提高应用程序的并发性能，但需要应用程序频繁轮询I/O状态，可能会导致CPU资源浪费。

I/O多路复用（I/O Multiplexing）：I/O多路复用允许单个线程或进程同时监听多个I/O事件的就绪状态。通过select、poll或epoll等机制，应用程序可以同时监听多个I/O操作，当有就绪的I/O事件时，再进行实际的读写操作。I/O多路复用可以提高并发处理能力，减少线程或进程的使用，适用于处理大量并发连接的场景。

信号驱动式I/O（Signal-Driven I/O）：信号驱动式I/O允许应用程序在I/O操作完成时接收一个信号通知。应用程序可以在发起I/O操作前设置信号处理器，并在操作完成后通过信号通知来处理相应逻辑。信号驱动式I/O兼具非阻塞和异步I/O的特性，可以提高并发性能。

异步I/O（Asynchronous I/O）：在异步I/O中，应用程序发起一个I/O操作后，可以立即返回继续执行其他任务，而无需等待操作完成。当I/O操作完成时，系统会通知应用程序，执行相应的回调处理。异步I/O将I/O操作的处理逻辑分离，可以有效地避免阻塞和轮询带来的性能问题，适用于高并发和高吞吐量的应用场景。异步I/O需要操作系统和应用程序的支持。

信号驱动式I/O和异步I/O的区别：
- ✦技术实现：信号驱动式I/O基于信号和信号处理器的机制实现，而异步I/O基于系统调用和回调函数实现。
- ✦通知方式：信号驱动式I/O通过发送信号来通知应用程序，而异步I/O通过回调函数来通知应用程序。
- ✦应用程序接口：信号驱动式I/O需要应用程序显式地设置信号处理器和关联到特定的I/O操作上，而异步I/O需要应用程序设置回调函数，并通过注册到操作系统的事件机制来进行回调通知。
- ✦可控性：信号驱动式I/O具有更低一级的控制，应用程序可以在信号处理器中自行处理操作完成的事件和结果。而异步I/O的控制由操作系统来处理，应用程序只需要定义回调函数来处理已完成的操作。

49. 网络编程你有实践过吗？如何保持一个连接是活跃的

(❌，不了解)

心跳机制（Heartbeat Mechanism）：通过定期发送心跳包，可以保持连接的活跃性。心跳包是一种特殊的数据包，用于向对方发送信号以表明连接依然有效。接收方收到心跳包后可以回复一个确认包，或者通过超时或其他机制来判断连接是否仍然可用。

1.传输层心跳机制：

✦TCP保活定时器（TCP Keepalive Timer）：TCP协议提供了一种保活定时器机制，可以在连接空闲一段时间后发送探测数据包来检测连接的活跃性。这个定时器可以通过设置TCP的SO_KEEPALIVE选项来启用，并设置相关的参数，如探测间隔、探测次数等。

2.应用层心跳机制：

✦应用层心跳检测：在应用层中定义心跳检测机制，通过定期发送特定的应用层控制消息，来检测连接的活跃性。接收方可以根据接收到的消息来判断连接是否仍然可用，以及执行相应的处理逻辑。

50. 你项目中的接口是怎么设计的？是什么样的格式?

Restful风格

51. 接口返回的数据是什么格式?

大多是JSON

52. AK、SK，sk是怎么算出来的?

MD5

追问：还了解别的加密算法吗?

不了解

53. 解释一下对称加密和非对称加密

依据加密后能不能推回来

追问：MD5属于哪种

非对称(❌乱讲的)