1. 项目用到 nginx，介绍一下。
反向代理，7*24 小时稳定热部署，主从备份，高并发，静态资源性能优于 Tomcat
2. 授权是怎么实现的，怎么和用户信息交互？
用户信息验证（md5 加密），结合 Token（jwt+RSA）
3. 密码为什么加盐
盐 bai 被称作“Salt 值”，这个值是 du 由系统随机生成的，并且只有系统知道。即 zhi 便两个用户使 dao 用了同一个密码，由于系统为它们生成的 salt 值不同，散列值也是不同的。
MD5 算法：生成一个 128 位散列值。
目的：防止反向查询密码，多次尝试破解。
4. 线程安全的实现

加锁：synchronized 或者 ReentrantLock

volatile 修饰变量

原子类

线程安全的容器

ThreadLocal

5. 两种锁的区别
目前性能优化后无差异，都是可重入锁
ReentrantLock ：1. 需要手动释放锁 2. 只可以修饰代码块 3. 底层调用 park 和 unpark 加锁。
优点：1. 可实现公平锁 2. 等待可中断 3. 可以设定多个条件
6. 写一个懒汉式单例模式
public class DCLSingleton { //Double Check Lock 单例模式双重校验 private static DCLSingleton instance = null; private DCLSingleton() { } public static DCLSingleton getInstance() { if (instance == null) {// 第一层判断主要是为了避免不必要的同步 synchronized (DCLSingleton.class) { if (instance == null) {//第二层判空是为了在 null 情况下创建实例 instance = new DCLSingleton(); } } } return instance; } } 写时候的问题：synchronized 里应该写类.class。
为什么不能写 this？ 一个类的 static 方***先于类的任何一个对象之前初始化，而 this 是当前对象。
7. 为什么用 volatile，会有什么问题？ 可不可以两次判断，有什么作用
加上 volatile，single = new Singleton(); 就可以保证 instance 对象每次都是从主内存中读取的，就可以采用 DCL 来完成单例模式了。当然，volatile 或多或少会影响到性能，但考虑到程序的正确性，牺牲点性能还是值得的。
双重判空，避免每次都进入同步状态。

推荐使用枚举类：

```
public class SingletonObject7 {     private SingletonObject7(){ }     枚举类型是线程安全的，并且只会装载一次     private enum Singleton{     NSTANCE;     private final SingletonObject7 instance;     Singleton(){     instance = new SingletonObject7();     }     private SingletonObject7 getInstance(){     return instance;     }     }     public static SingletonObject7 getInstance(){     return Singleton.INSTANCE.getInstance();     } }
```

8. 为什么用枚举类？

避免反射、克隆、序列化等破坏单例模式。

直接解决：反射：在私有构造方法中加入判断，阻止新实例生成

序列化：不实现序列化，或者重写序列化，重写 clone()

9. 分布式情况下访问数据库的线程安全问题

使用分布式锁：redis、zookeeper、数据库锁（乐观锁和悲观锁：写锁）

使用异步队列讲多线程转为单线程操作

10. 分布式事务与分布式锁的区别

事务是完成 ACID 的。锁是保障资源访问线程安全的。

11. 线程的状态

新建 就绪 运行 死亡 阻塞 等待 长等待

12. 等待和阻塞的区别，怎么进入等待？

阻塞：当一个线程试图获 bai 取一个内部的对象锁，而该锁 du 被其他线程持有时，则该线程进入阻塞状态。 等待：当一个线程等待另一个线程通知调度器一个条件时，该线程进入等待状态。例如调用：Object.wait()、Thread.join()以及等待 Lock 或 Condition。

13. 用过线程池么？为什么用线程池（优点）？