【自我介绍】

**面试官**您好，我叫**胡景峰**。

**本科**就读于 合肥工业大学 计算机学院

**硕士**就读于 东南大学 软件学院（明年6月毕业）

本科和硕士阶段的 **课程成绩排名** 均为前5%

硕士组内是以 **虚拟现实与人机交互** 为大方向

我的科研方向主要是 **渲染+边缘计算** 做一些优化

个人技术栈以**Java**为主，熟悉Spring、MySQL、Redis等

平时有写文档或博客**反思总结**的习惯

是一个**长期主义**者，相信 微小而正确的习惯 带来的 复利效应

【英文介绍】

Hello, my name is Jingfeng Hu.

I **completed undergraduate studies** in Computer Science at Hefei University of Technology.

I’m currently **pursuing Master's degree** in Software Engineering at Southeast University. （and will graduate in 2026.）

I have earned **several honors**, including:

* National Scholarship
* Title of Excellent Graduate

and prizes from competitions like

* China Software Cup
* Mathematical Modeling Contest

I usually use **Java** and I’ve learned about:

* Concurrency Mechanisms
* TCP/IP and HTTP protocols
* MySQL and Redis

I **value teamwork**, **enjoy challenges**, and am committed to **continuous learning and improvement**. I believe my **technical skills and positive attitude** will allow me to **quickly adapt and contribute effectively to our team**.

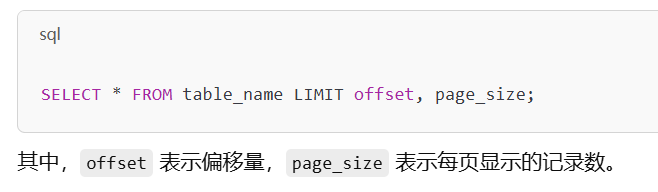
【项目经历】

吃了么外卖平台

这是一个**网络教程项目**，主要想通过项目实战**理解并应用Spring**。

分页查询

使用 **LIMIT 子句（偏移量、每页记录数）**实现分页查询：



深度分页查询（分页到1k页2k页）的问题

1. **性能问题**：  
   LIMIT 子句是先读取、再筛选的，随着页数增加，偏移量增大，数据库需要扫描大量的数据页才能定位到指定页的数据。  
   例如，每页 10 条、1000页，偏移量就是 10000，数据库需要先扫描前面 10000 条记录，这会消耗大量的时间和资源。
2. **内存占用问题**：  
   深度分页查询需要获取大量数据，可能使**临时结果集占用过多内存**。若内存不足，数据库可能会使用磁盘临时文件来存储数据，这会导致查询性能大幅下降，因为磁盘 I/O 操作比内存操作慢得多。

深度分页查询（分页到1k页2k页）的优化方案

1. 使用**覆盖索引**进行分页查询，避免回表操作。  
   例如，如果查询只需要 id、name 列，且这两列在一个联合索引中，那么可以创建覆盖索引来优化查询。  
   分页查询中如何避免全表扫描：
   1. 避免使用 SELECT \*
   2. 避免在查询条件中对字段进行函数操作
   3. 创建合适的索引
2. 使用上一页的最后一条记录的主键ID作为下一页查询的条件，通过 WHERE 子句过滤数据。  
   例如，假设主键是 id，上一页最后一条记录的 id 是 100，每页显示 10 条记录，那么下一页的查询语句可以是 SELECT \* FROM your\_table\_name WHERE id > 100 ORDER BY id LIMIT 10。这样可以避免使用大的偏移量，提高查询性能。
3. **优化**数据库的缓存大小、应用层Redis**缓存** 、分库分表减少单表数据量

MyBatis Plus 提供了 Page 类用于分页查询



公共字段自动填充

通过自定义注解和AOP切面编程，实现了在插入和更新操作时自动填充**创建时间、更新时间、创建人、更新人**等字段。

MyBatis-Plus中，

1. 在实体类的相应字段上添加@TableField注解，并设置fill属性。
2. 实现MetaObjectHandler接口，重写insertFill和updateFill方法，设置字段的自动填充值。



如何获取当前登录用户的ID用于填充创建人和更新人字段？

在用户登录成功后，将用户ID存入ThreadLocal中，之后在MetaObjectHandler的实现类中，通过ThreadLocal获取当前用户ID，并设置到createUser和updateUser字段中。



在《苍穹外卖》项目中，如何实现公共字段的自动填充？

通过自定义注解和切面编程实现。具体步骤如下：

1. 定义自定义注解@AutoFill，用于标识需要自动填充的Mapper方法。
2. 创建切面类AutoFillAspect，在方法执行前，获取方法参数并通过反射设置公共字段的值。
3. 在需要自动填充的Mapper方法上添加@AutoFill注解。

公共字段自动填充可能遇到哪些问题？如何解决？

**问题**：在更新操作中，公共字段未被自动填充。

**解决方案**：确保在实体类中，字段的@TableField注解设置了正确的fill属性，如FieldFill.INSERT\_UPDATE。

**问题**：无法获取当前用户ID。

**解决方案**：使用ThreadLocal在用户登录时保存用户ID，并在自动填充时获取。

**问题**：使用自定义注解和切面时，反射设置字段失败。

**解决方案**：确保实体类中存在对应的setter方法，并且方法名与反射调用一致。

订单状态定时处理

1. **支付超时订单的自动取消**：如果用户下单后超过15分钟仍未支付，系统会自动将订单状态修改为“已取消”。
2. **派送中订单的自动完成**：对于状态为“派送中”的订单，系统会在每天凌晨1点检查，如果订单处于该状态超过一定时间（例如60分钟），则自动将其状态修改为“已完成”。

使用Spring Task实现定时任务

通过Spring框架提供的任务调度工具**Spring Task**实现，使用**@Scheduled**注解结合**Cron表达式**定义定时任务。

在Spring Boot中，可以使用Spring Task实现定时任务

1. 在启动类上添加@EnableScheduling注解，开启定时任务功能。
2. 创建一个任务类，并添加@Component注解，使其被Spring容器管理。
3. 在任务方法上添加@Scheduled注解，定义任务的执行时间。



如何处理支付超时的订单？

1. 定义一个定时任务，每分钟执行一次。
2. 在任务中查询状态为“待支付”且下单时间早于当前时间减去15分钟的订单。
3. 将这些订单的状态更新为“已取消”，并记录取消原因和取消时间。



如何处理长时间处于“派送中”状态的订单？

1. 定义一个定时任务，每天凌晨1点执行。
2. 在任务中查询状态为“派送中”且派送开始时间早于当前时间减去60分钟的订单。
3. 将这些订单的状态更新为“已完成”。



对菜品信息进行缓存、数据一致性问题、Redis

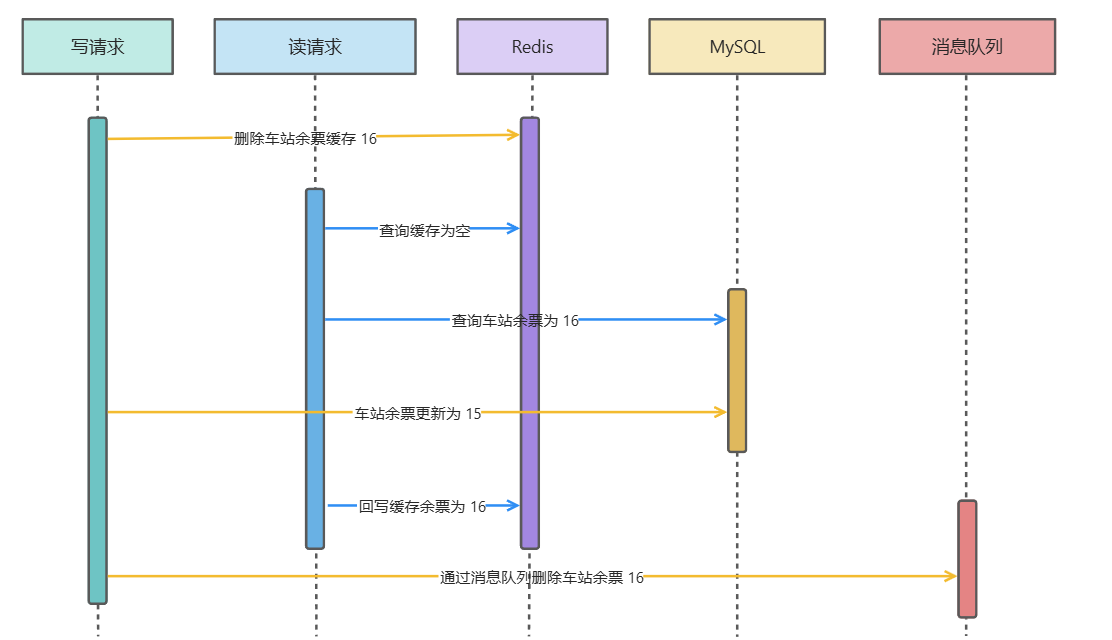
**方案1：先写缓存、再写数据库（×）**

**方案2：先写数据库、再写缓存（×）**

**方案3：先删缓存、再写数据库（×）**

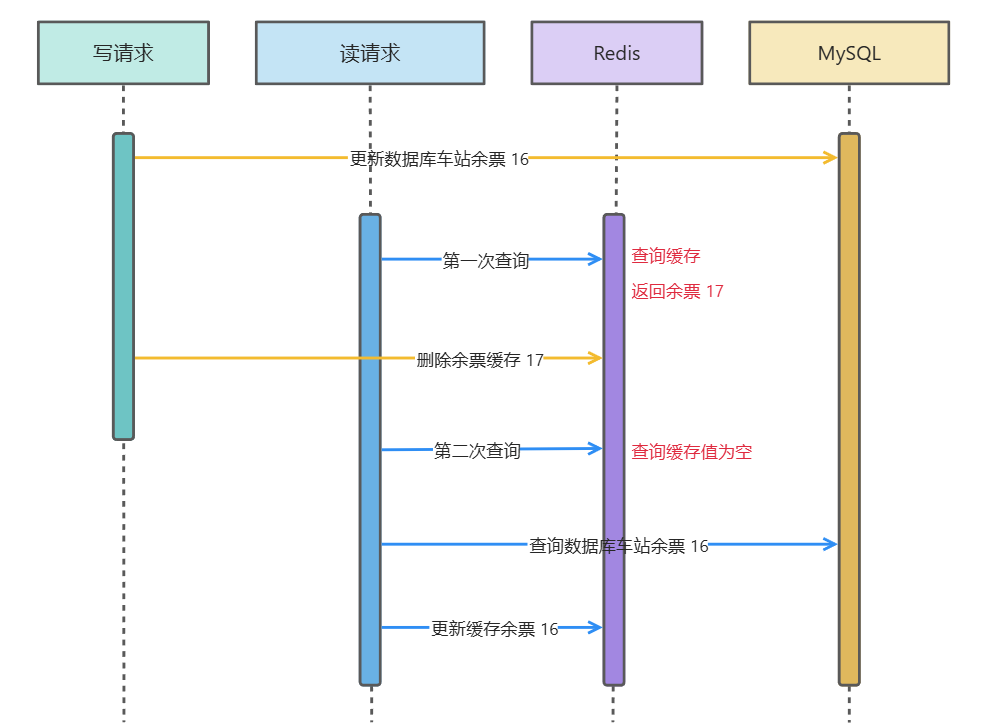
**方案4：先删缓存再写数据库再删缓存（缓存双删）**

写数据库后、二删缓存前，写数据库前的查询刚好回写了旧数据到缓存



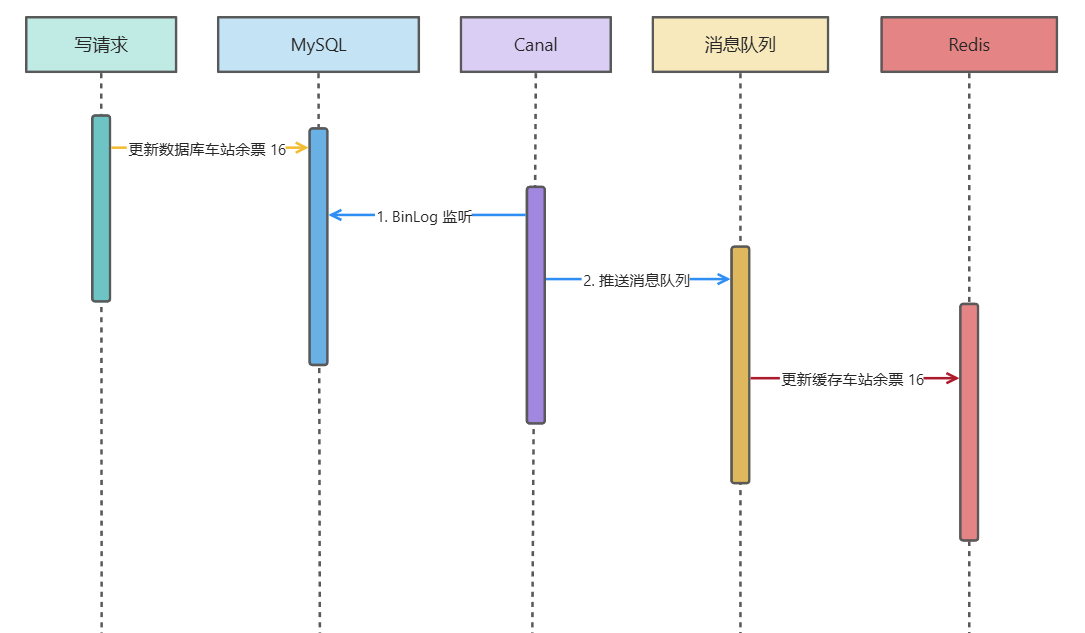
**方案5：先写数据库再删缓存**

写数据库后、删缓存前，缓存里的数据不是最新的，此时被查询会不一致



**方案6：先写数据库再通过Binlog异步更新缓存**

理由：高并发场景删除缓存可能并不合适



Spring IoC、AOP 思想

🍃Spring IoC

控制反转IoC是**思想**，将**管理对象**的控制权从**代码**转移到**容器**。

依赖注入DI

依赖注入DI是实现方式，容器根据配置将所需依赖注入对象。

依赖注入的3种方式，按推荐从高到低，分别是：

1. 构造器注入（推荐）：适用于强制依赖的注入
2. Setter注入：适用于可选依赖或个人开发的模块的注入
3. @Autowired字段注入（不推荐）：因为无法通过构造器或Setter方法修改依赖关系，降低了代码的可维护性和可测试性。

接口回调注入：如BeanFactoryAware，适用于获取Spring容器对象、动态加载Bean。

Bean的配置

**Spring Bean**就是 **Spring 容器管理的 Java 对象**

注册到容器

1. XML
2. @Import将普通类注册到容器中
3. @Component + @ComponentScan：类级别、自动扫描、自定义

* **@Component**：通用。
* **@Controller**：控制层
* **@Service**：服务层
* **@Repository**：持久层。异常转换：将低层次的数据库异常（如 SQLException）转换为 Spring 统一的 DataAccessException。

1. @Bean + @Configuration：方法级别、手动注册、用于第三方无法修改的类

@Autowired 和 @Resource 的区别

都是**自动装配Bean**的注解

* **@Autowired：**Spring 特有、默认**按类型**、即没有匹配到时注入 null不抛异常、支持结合 @Qualifier 注解指定注入的 Bean 名称。
* **@Resource：** JSR-250 标准（更适合标准化和跨框架）、默认**按名称**、如果未指定名称且没有匹配的 Bean会按类型注入。

@Qualifier 和 @Primary 的区别

都是**解决 Bean 注入时的歧义问题**的注解

* **@Qualifier**：指定**名称**选择对应的实现 Bean。
* **@Primary** ：指定**默认**注入哪个 Bean。

注：@Qualifier 可以覆盖 @Primary 的默认行为。

条件配置

**@Profile("dev") 指定环境**，比如 dev 开发、prod 生产

**@Conditional条件加载**

* @Conditional**OnProperty** 根据配置文件属性值，常用于启用或禁用模块
* @Conditional**OnMissingBean** 避免重复注册

Bean 的作用域

1. **singleton（默认）：** 单例bean，适用于无状态的共享资源。
2. **prototype：**适用于短期使用的、有状态、非线程安全的对象。

* 唯一标识符（UUID 、验证码、Token）
* 临时数据（表单数据缓存、文件上传临时存储）

1. **request：** 每个 HTTP 请求创建一个实例，如表单数据处理。
2. **session：** 每个会话创建一个实例，如用户信息缓存。
3. **application/global-session：** 在Web应用启动时创建一个实例，如统计数据。
4. **websocket：** 每个 WebSocket 会话创建一个实例，适用于WebSocket 连接状态管理。

作用域与生命周期的关系

* **singleton**：Bean 的生命周期与容器一致。
* **prototype**：容器只负责创建，不管理、不销毁，客户端决定何时销毁。
* 其他：Bean 的生命周期分别与请求、会话、应用或 WebSocket一致。

有状态的单例bean是非线程安全的

1. 如果用singleton ，避免使用可变状态；  
   如果需要可变状态，使用prototype作用域；
2. 加锁；  
   使用ThreadLocal保存变量；

Bean 的生命周期

1. 容器通过**反射**根据**配置文件**实例化 Bean
2. **依赖注入**
3. **初始化**
   1. **Aware注入**  
      获取BeanFactory、ApplicationContext 等容器资源
      * BeanNameAware
      * BeanClassLoaderAware
      * BeanFactoryAware
   2. **BeanPostProcessor#postProcessBeforeInitialization**（应用：创建动态代理，实现AOP）
   3. Initialization（应用：缓存初始化）
      * + @PostConstruct
        + InitializingBean#afterPropertiesSet
        + initMethod
   4. **BeanPostProcessor#postProcessAfterInitialization**
4. **使用**
5. **销毁**@PreDestroy调用DisposableBean#destroy。  
   资源释放（关闭数据库连接、文件句柄、线程池）  
   会话管理（清理用户会话或缓存）   
   注意：容器关闭时才调用销毁逻辑所以看不到，可以destroy-method执行：
   1. 暴力手动关闭：ClassPathXmlApplication#close
   2. 注册关闭钩子：ClassPathXmlApplication#registerShutdownHook()

🍃Spring AOP

AOP核心思想

AOP（Aspect-Oriented Programming，面向切面编程）

通用功能模块化（权限、日志、事务、性能），避免代码重复。

通过代理的方式，先拦截穿插，再调用真正方法。

AOP相关术语

**切面(Aspect) = 切点(Pointcut) + 通知(Advice)**

* **切面（Aspect）：**封装了通用功能（权限、日志、事务、性能）
* **切点（Pointcut）：**特殊的连接点，通过切点表达式找到想要织入的方法
  + **目标（Target）：** 被切面增强的对象
  + **连接点（JoinPoint）：** 目标对象的所属类的所有方法均为连接点
* **通知（Advice）：** 拦截到目标对象的连接点之后**要执行的操作**
  + **代理（Proxy）：** 代理对象包含目标对象的原始方法和增强逻辑
  + **织入（Weaving）：** 将切面应用到目标对象上的过程，Spring AOP是运行时织入

常见的通知类型

* **前置通知（Before）**
* **后置通知（After）**
* **环绕通知（Around）**
* **异常通知（AfterThrowing）**
* **返回通知（AfterReturning）**

Spring4版本

1. 环绕前通知
2. 前置通知Before
3. 被增强的方法
4. **环绕后通知 / 无（异常时）**
5. 后置通知After
6. **AfterReturning 返回通知 / AfterThrowing 异常通知（异常时）**

Spring5版本（汉堡包）

1. 环绕前通知
2. 前置通知Before
3. 被增强的方法
4. **AfterReturning 返回通知 / AfterThrowing 异常通知（异常时）**
5. 后置通知After
6. **环绕后通知 / 无（异常时）**

AOP工作机制

JDK动态代理和CGLIB 动态代理的区别

**JDK 动态代理**

* SpringFramework的默认实现
* 基于接口实现，通过 Java 的反射机制实现
* 适合实现接口的的类，没有接口的话会有报错

**CGLIB 动态代理**

* SpringBoot2的默认实现
* 基于类继承，通过ASM 字节码生成工具生成继承目标类的子类。
* 适合没有接口的类
* 不能代理 final 类和 final 方法

Spring AOP 和 AspectJ AOP 的区别

**Spring AOP**：

* 基于动态代理，仅支持运行时AOP
* 更轻量、更方便，适合大部分业务场景，但仅适用于 Spring 容器管理的对象

**AspectJ AOP**：

* 支持编译时、类加载时、运行时AOP
* 更灵活、性能高，适合日志、监控

🍃Spring 循环依赖

什么是循环依赖

多个 Bean **循环引用**导致容器**无法正常初始化**它们。

例如：

* A 要依赖 B，发现 B 还没创建，于是开始创建 B
* 创建的过程发现 B 要依赖 A，但 A 还没创建好

通过三级缓存解决循环依赖

三级缓存

1. **一级缓存（Singleton Objects Map 单例对象）**：存放已经实例化、属性填充、初始化的 Bean。
2. **二级缓存（Early Singleton Objects Map 早期单例对象）**：存放已实例化，但未完成属性注入和初始化的 Bean。也就是三级缓存中ObjectFactory产生的对象，与三级缓存配合使用的，可以防止 AOP 的情况下，每次调用ObjectFactory.getObject()都是会产生新的代理对象的。
3. **三级缓存（Singleton Factories Map 单例工厂）**：存放对象工厂ObjectFactory，ObjectFactory的getObject()方法最终调用getEarlyBeanReference()方法创建早期Bean（特别是为了支持AOP代理对象的创建）。

Spring创建Bean流程

1. 当 Spring 创建 A 时发现 A 依赖了 B ，又去创建 B，B 依赖了 A ，又去创建 A；
2. 在 B 创建 A 的时候， A 此时还没有初始化完成，因此在 一二级缓存 中肯定没有 A；
3. 那么此时就去三级缓存中调用 getObject() 方法最终调用getEarlyBeanReference() 方法去生成并获取 A 的前期暴露的对象。
4. 然后就将这个 ObjectFactory 从三级缓存中移除，并且将前期暴露对象放入到二级缓存中，那么 B 就将它注入到依赖，来支持循环依赖。

只用两级缓存够不够

* 在没有 AOP 时，需要一级和三级缓存解决循环依赖。
* **在涉及AOP 时，二级缓存避免同一 Bean 有多个代理对象的问题。**
  + 直接使用二级缓存不做任何处理会导致我们拿到的 Bean 是未代理的原始对象。
  + 如果二级缓存内存放的都是代理对象，则违反了 Bean 的生命周期。
* getEarlyBeanReference会判断这个对象是否需要代理，如果否则直接返回，如果是则返回代理对象。

ObjectFactory 的使用场景

* **懒加载 Bean**：当某个 Bean 的创建过程可能耗时较长、依赖的资源较重、或需要在运行时决定动态 Bean 使用时，可以通过 ObjectFactory 进行懒加载，避免容器启动时不必要的 Bean 创建。
* **避免循环依赖**：在某些情况下，两个 Bean 可能相互依赖，导致循环依赖问题。通过使用 ObjectFactory，可以延迟其中一个 Bean 的创建，避免循环依赖。在 Spring 的循环依赖的三级缓存的 map 里面存储的就是 ObjectFactory，用于延迟代理对象的创建。

ObjectFactory和Provider 的区别

都能够惰性获取 Bean 实例

* ObjectFactory 是 Spring 内部的接口。
* Provider 则是 Java 标准中的一部分，适用于更通用的场景。

通过@Lazy解决循环依赖

* 1. A 的构造器上添加 @Lazy 注解之后，延迟 B 的实例化，不会立即创建 B，而是会注入一个 B 的代理对象。
  2. 由于此时 B 仍未被真正初始化，A 的初始化可以顺利完成。
  3. 等到 A 实例实际调用 B 的方法时，代理对象才会触发 B 的真正初始化，在注入 B 中的 A 属性时，此时 A 已经创建完毕了，就可以将 A 给注入进去。

12306 铁路购票系统

GitHub找的**开源项目二开**，主要想对**高并发**有一个实践和应对。

通过复合分片算法完成订单数据分库分表，支持订单号和用户查询

订单分库分表 选择**订单号**和**用户ID**作为复合分片键。

采用基因算法，将用户ID的哈希值嵌入到订单号中。

采用映射表将用户名/手机/邮箱与用户ID建立映射关系，存储在单独的索引表中。登录时先通过索引表查询到用户ID，再根据用户ID定位到具体的分片，从而避免全表扫描。

分库分表后选择雪花算法作为ID（有序、唯一、性能）

* 时间戳 + 机器ID + 序列号

封装缓存组件库，避免注册时用户名全局唯一带来的缓存穿透问题

缓存穿透定义：

请求不存在于数据库的数据 造成 数据库负载、缓存内存耗尽、用户体验差

尝试解决方案：

1. 对不存在的 Key 缓存并把值设为Null，设置短暂过期时间（如 60 秒）。  
   **缺点：**尝试但没注册一个不存在的用户名，该值60s内都不可被注册
2. 查询缓存不命中时使用分布式锁来保证只有一个线程访问数据库。  
   **缺点：**其他用户注册请求缓慢或超时
3. 布隆过滤器存已注册用户名，不在布隆过滤器的一定不存在-可用，如果用户名在布隆过滤器中再查询缓存或数据库，。  
   **缺点：**布隆过滤器不能删除元素，注销的用户名无法再次使用。
4. Redis Set存已注册用户名，检查是否在集合内。  
   **缺点：**占用内存。

最终解决方案：

布隆过滤器 + Redis Set缓存

1. 布隆过滤器不存在，说明数据库没有-可用

2. Redis Set缓存存在，说明已注销-可用

3. 查询数据库到底有没有

使用 BinLog 配合 RocketMQ 消息队列完成 MySQL 数据库与 Redis 缓

存之间的数据最终一致性。

**方案1：先写缓存、再写数据库（×）**

**方案2：先写数据库、再写缓存（×）**

**方案3：先删缓存、再写数据库（×）**

**方案4：先删缓存再写数据库再删缓存（缓存双删）**

写数据库后、二删缓存前，写数据库前的查询刚好回写了旧数据到缓存

**方案5：先写数据库再删缓存**

写数据库后、删缓存前，缓存里的数据不是最新的，此时被查询会不一致

**方案6：先写数据库再通过Binlog异步更新缓存**

理由：高并发场景删除缓存可能并不合适

通过 RocketMQ 延时消息实现购票十分钟未支付情况下取消订单功能

通过 RocketMQ 延时消息特性，完成用户购票 10 分钟后未支付情况下取消订单功能。

* [**订单延时关闭功能技术选型**](https://www.yuque.com/magestack/12306/sazb1bn2a1okwtlu)（最后选择RocketMQ实现）
* [**创建订单并支付后延时关闭订单消息怎么办？**](https://www.yuque.com/magestack/12306/ldw8nxp96yfg7cgx)
  + [手摸手之消息队列正确使用姿势](https://www.yuque.com/magestack/12306/dhm3lr598gtt23od)（看不大懂，要补）
* 1. 如何取消十分钟未支付的订单？

**RocketMQ延时消息**：

* 订单生成时，向 RocketMQ 发送一条延时消息，设置延时时间为十分钟。
* 十分钟后，RocketMQ 将消息投递给消费者，检查订单状态。
* 如果订单状态仍为未支付，则释放库存并将订单状态标记为已取消。
* 2. 如何避免已支付错误取消？

在消费者处理延时消息时进行**状态校验**：消费者接收到延时消息后，首先检查订单的支付状态。如果订单已支付，则直接忽略消息，不执行取消操作。

订单延时关闭功能技术选型

1. **定时任务（xxl-job、PowerJob、shardingsphere-elasticjob）**
   1. **延时精度**
   2. **高并发场景下，可能导致大量的定时任务同时执行，造成系统负载过大。**
   3. **分库分表问题：订单表按照用户标识和订单号进行了分库分表，那这样的话，和上面说的根据订单创建时间去扫描一批订单进行关闭，自然就行不通。因为根据创建时间查询没有携带分片键，存在读扩散问题。**
2. RabbitMQ
   1. 延时精度：基于消息的 TTL（Time-To-Live）来实现的，因此消息的延时时间并不是完全准确的，可能会有一定的误差。
   2. 高并发问题：消息堆积
   3. 重复消息问题：由于网络原因或其他不可预知的因素，可能会导致消息重复发送到订单队列。如果没有处理好消息的幂等性，可能会导致订单重复关闭的问题，从而造成数据不一致或其他异常情况。
3. Redis 过期监听
   1. 延时精度：Redis 的过期时间是通过定时器实现的
   2. Redis 宕机或重启，那些已经设置了过期时间但还未过期的订单信息将会丢失，导致这部分订单无法正确关闭  
      Redis 5.0 之前是不保证延迟消息持久化的，如果客户端消费过程中宕机或者重启，这个消息不会重复投递。5.0 之后推出了 Stream 功能，有了持久化等比较完善的延迟消息功能。
4. Redisson 的 RDelayedQueue 功能  
   基本上 Redis 过期监听消息存在的问题，RDelayedQueue 也都会有，因为 RDelayedQueue 本质上也是依赖 Redis 实现。
5. RocketMQ  
   时间轮算法  
   在订单生成时，我们将订单关闭消息发送到 RocketMQ，并设置消息的延迟时间为十分钟。RocketMQ 支持设置消息的延迟时间，可以通过设置消息的 delayLevel 来指定延迟级别，每个级别对应一种延迟时间。这样，订单关闭消息将在十分钟后自动被消费者接收到。  
   rocketmq好整合一点，性能更高。rabbitmq是一个大型中间件，光docker部署就上1个G了。而且还封闭，性能比不过rocketmq。

【专业技能】

熟悉 Java 集合、I/O 流、异常、反射等机制，阅读过 HashMap 源码，了解其扩容机制

熟悉 Java 并发容器和锁，如 ConcurrentHashMap、CAS、AQS 等

熟悉 SpringBoot 和 SSM 框架，能够运用并开发项目，熟悉 AOP、IOC 等原理

熟悉 MySQL 及其索引、事务、日志、锁等机制，了解读写分离、分库分表

熟悉 Redis 及其线程模型、持久化机制，熟悉缓存穿透、击穿、雪崩解决方案

熟悉 TCP/IP 网络模型以及 TCP、UDP 等协议，对 HTTP 协议的版本变化有所研究

熟悉操作系统及其进程、线程、虚拟内存等概念，熟悉进程间通信、多路复用等技术

熟悉 Linux 常用命令（如 ps、top、netstat、tail、kill），熟悉 grep、awk、sed 三剑客

【荣誉奖项】

“中国软件杯”大学生软件设计大赛国家三等奖

* 基于金蝶云低代码平台这一完全陌生的技术栈，通过阅读平台文档短时间内零基础学习开发完成图书管理系统
* 锻炼了在未知领域中快速学习与落地的能力
* 培养了协调资源、带领团队的组织能力

全国大学生数学建模竞赛省二等奖

* 与队友连续奋战三天三夜，从问题建模、算法实现到论文撰写，每一个环节都精雕细琢。过程中我深刻体会到持续专注推进一件复杂任务所带来的成就感。
* 锤炼了数理逻辑与编程建模能力

提升了在高压下保持思维清晰、推进任务执行的能力

【反问环节】

* 能谈一谈您作为一个公司老员工对公司的感受，和其他公司不同之处？
* 我觉得我这次表现不是很好，您可以给一些评价或建议吗？
* 这个岗位为什么还在招人？
* 实习生主要工作？